VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GRAFICKÉ DEMO S VESMÍRNOU TEMATIKOU

GRAPHICS DEMO WITH A SPACE THEME

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MARTIN PRAJKA**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2018

Zadání bakalářské práce/21168/2017/xprajk00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Prajka Martin**
Obor: Informační technologie
Téma: **Grafické demo s vesmírnou tematikou**
Graphics Demo with a Space Theme
Kategorie: Počítačová grafika

Pokyny:

1. Seznamte se s problematikou vykreslování v reálném čase pomocí OpenGL.
2. Vyberte a popište několik technik vykreslování a animace vhodných pro demo s vesmírnou tematikou.
3. Implementujte vybrané techniky a vyladťe jejich vizuální kvalitu a efektivitu vykreslování.
4. Integrujte implementované techniky do uceleného grafického dema.
5. Testujte vytvořené demo na uživatelích/divácích a vylepšujte je k dokonalosti.
6. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- John Kessenich, Graham Sellers, Dave Shreiner: OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V (9th Edition), The Khronos OpenGL ARB Working Group, 2016
- Anton Gerdelan: Anton's OpenGL 4 Tutorials, 2014
- série knih: Game Programming Gems
- série knih: GPU Gems

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3, značné rozpracování bodu 4.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Herout Adam, prof. Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
L.S. 612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato bakalářská práce se zabývá tvorbou grafického dema bez omezení velikosti za použití OpenGL. Práce implementuje post-processing efekty, jako jsou shadow-mapping, bloom a gaussian blur. Dále popisuje částicové systémy a animaci kamery. Demo se odehrává ve vesmírném prostředí a je rozděleno do tří scén. Konec práce je zaměřen na zhodnocení výsledků uživatelských a výkonových testů.

Abstract

This bachelor's thesis deals with the creation of graphics demo with unlimited size by using OpenGL. The thesis implements post-processing effects such as shadow mapping, bloom and gaussian blur. In addition, it describes particle system and camera animation. The demo is placed in space and it is separated into the three scenes. The end of the thesis is focused on the results of user's and performance tests.

Klíčová slova

Počítačová grafika, OpenGL, Grafické demo, Částicové systémy, Efekty, Post-processing, Vesmír, Exploze, Shadow mapping, Bloom, Zvuky, Animace kamery

Keywords

Computer graphics, OpenGL, Graphics demo, Particle system, Effects, Post-processing, Space, Explosion, Shadow mapping, Bloom, Sounds, Camera animation

Citace

PRAJKA, Martin. *Grafické demo s vesmírnou tematikou.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Grafické demo s vesmírnou tematikou

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. Ing. Adama Herouta Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Prajka
15. května 2018

Poděkování

Rád bych poděkoval prof. Ing. Adamu Heroutovi, Ph.D za odborné rady a připomínky k aplikaci i dokumentaci, dále Ing. Tomáši Miletovi za odbornou pomoc při řešení artefaktů v shadow mappingu, slečně Kateřině Rozehnalové za gramatickou korekci práce a svojí mamce za zprostředkování uživatelských testů u nich ve firmě.

Obsah

1 Úvod **2**

2 Úvod do problematiky **3**
2.1 Demo . . . 3
2.2 Grafika zobrazovaná v reálném čase . . . 3
2.3 Téma . . . 4

3 Grafické techniky **5**
3.1 Shadow mapping . . . 5
3.2 Rozmazání obrazu Gaussovou funkcí . . . 8
3.3 Bloom . . . 9
3.4 Phongův osvětlovací model . . . 11
3.5 Mlha . . . 15
3.6 Tvorba terénu pomocí výškové mapy . . . 16
3.7 Částicové systémy . . . 20

4 Návrh a Implementace **21**
4.1 Použité knihovny . . . 21
4.2 Implementační detaily . . . 22
4.3 Antialiasing . . . 24
4.4 Animace . . . 25

5 Testy a výsledky **28**
5.1 Uživatelské testy . . . 28
5.2 Výkonové testy . . . 29

6 Závěr **32**
6.1 Možná rozšíření . . . 32

Literatura **33**

A Obsah přiloženého paměťového média **34**

B Konfigurační soubor **35**

C Plakát **36**

D Testovací dotazník **37**

Kapitola 1

Úvod

Cílem práce je vytvořit grafické demo bez omezení velikosti zasazené do vesmírného prostředí s efekty akcelerovanými na grafické kartě. Vesmírné prostřední jsem zvolil, protože na internetu se nevyskytuje velké množství dem na toto téma. Práce je členěna do šesti kapitol s následujícím zaměřením. Kapitola 2 popisuje obecnou charakteristiku dema a přibližuje jeho vývoj. Dále vysvětluje zobrazování v reálném čase, jeho využití a zmiňuje metodu z-buffer. Kapitola je ukončena seznámením s tématem vytvořeného dema. V kapitole 3 jsou podrobně popisovány jednotlivé použité grafické techniky a způsob, jakým byly aplikovány v demu, kterým se tato práce zabývá. Dále se práce věnuje návrhu a implementaci v kapitole 4, kde popisuje programovací principy, zvolený jazyk pro psaní aplikace, poznatky z implementace grafických technik, které se odlišují od obecných definic, podrobnější popis částicového systému, dále použité knihovny a animaci kamery. Kapitola 5 je zaměřena na uživatelské a výkonové testování. Výsledky výkonových testů jsou zobrazeny graficky. V poslední kapitole 6 je práce shrnuta a je zde popsáno její možné rozšíření a pokračování.

Kapitola 2

Úvod do problematiky

2.1 Demo

Grafická dema [5] jsou programy založené na 3D animacích, které mohou využívat i 2D efekty, obvykle obsahují hudbu nebo zvuky a nemívají interakci s uživatelem. Hlavním rozdílem mezi demem a animací je, že demo se počítá a vykresluje v reálném čase (Sekce 2.2), z čehož vyplývá, že by mělo využívat efektivně grafický hardware.

Demoscéna byla inspirována technikami využívanými především v počítačových hrách. Její počátky jsou připisovány počítačovým crackerům, kteří přidávali svá dema (intra) před nelegálně šířený software jako jejich podpis. Poté začaly vznikat i soutěže mezi vývojáři dem a v dnešní době může být demo chápáno jako určitý druh umění.

Dema se obvykle tvoří jako intra s omezenou velikostí, nejčastěji do 64kB, tato intra nevyužívají externích souborů pro modely a textury. Nicméně mohou existovat i dema s neomezenou velikostí, které využívají externích souborů pro modely, textury a zvuky. Od dem tohoto typu se očekává kvalitnější vizuální zážitek. Právě do této kategorie spadá demo, jímž se práce zabývá.

2.2 Grafika zobrazovaná v reálném čase

Zobrazování v reálném čase je část grafiky, která se zabývá tvořením a analýzou grafických entit v reálném čase. Jak se zmiňuje v knize Real-time rendering [4] v první kapitole, jedná se o nejvíce interaktivní oblast z počítačové grafiky. Zobrazování v reálném čase klade důraz na rychlost vykreslování tak, aby uživatel nevnímal přepínání mezi snímky, ale spíše vnímal, že je součástí dynamického procesu. Rychlost zobrazování snímků se měří ve snímcích za sekundu (*frame per second*, fps) nebo také v Hertzích (Hz). Minimální hranice, aby bylo možné o aplikaci prohlásit, že běží v reálném čase, je okolo 15 fps.

Grafika zobrazovaná v reálném čase je spojována s trojrozměrnou grafikou, která je obvykle akcelerována na grafické kartě (*graphic processing unit*, GPU). Hardwarová akcelerace ale nemusí být podmínkou. V minulých letech mohla být 3D grafika implementována i bez grafických akcelerátorů. Avšak s čím dál větší dostupností grafických karet se GPU staly důležitou komponentou pro spoustu aplikací využívajících zobrazování v reálném čase.

Vysoká rychlost zobrazování snímků se také zaručuje vhodným algoritmem pro řešení viditelnosti. V grafice zobrazované v reálném čase se nejčastěji využívá metoda z-buffer (někdy též depth buffer). V této metodě se každý objekt skládá z jednotlivých primitiv (nejčastěji trojúhelníků). Základem metody je paměť, která obsahuje Z souřadnici (hloubku) bodu,

který je nejblíže k pozorovateli v daném pixelu. Funguje na základě porovnání hloubky bodů různých objektů pro jeden pixel, přičemž nejbližší bod je uložen do paměti. Jedná se o rychlý algoritmus se snadnou realizací v hardwaru.

Grafiku zobrazovanou v reálném čase využívá především filmový a herní průmysl, ale může být využívána také pro simulátory činností různých profesí (např. piloti, vojáci) nebo ve vědě, kde je potřeba simulovat různé pokusy.

2.3 Téma

Téma grafického dema je pojato formou příběhu. Příběh popisuje cestu ze Země do vesmíru a následný průlet sluneční soustavou. První scéna je zasazena do prosluněného letního dopoledne v hornatých pláních u odletové rampy raketoplánu. Pohyb po scéně je udělán formou pohledu z první osoby, což má vyvolat dojem, že se jedná o kosmonauta, který se právě chystá na odlet. Před odletem si prochází prostory vesmírného střediska. Scéna je doprovázena zpěvem ptactva. Druhá scéna se odehrává v kabině raketoplánu, který právě startuje do kosmu. Raketoplán prolétá skrz mraky a je zde vidět, jak s výškou tmavne obloha. Atmosféru dokresluje zvuk motoru. Třetí, a zároveň poslední, scéna se odehrává v samotném vesmíru, kde je zobrazen průlet mezi některými planetami sluneční soustavy. Kamera prolétá i meteorovým rojem a u Slunce se poukazuje na sluneční erupci. Scéna je ukončena explozí asteroidu. V podkresu zní vesmírná hudba a zvuk exploze.

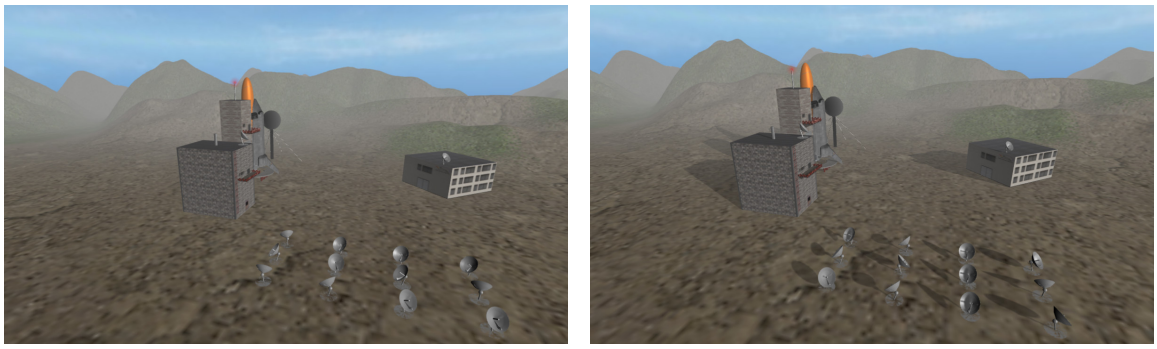
Kapitola 3

Grafické techniky

Tato kapitola popisuje všechny grafické techniky, které jsou implementovány v demu. Mezi ně patří shadow mapping, bloom, rozmazání obrazu pomocí Gaussovy funkce, Phongův osvětlovací model, efekt mlhy, tvorba terénu pomocí výškové mapy a částicový systém.

3.1 Shadow mapping

Stíny vznikají, pokud přímočaře šířící se světelné paprsky nedorazí na požadovaný objekt z důvodu zablokování jejich trajektorie jiným neprůhledným objektem. Stíny pomáhají pochopit 3D scénu a vzájemnou polohu mezi objekty (viz obrázek 3.1).



Obrázek 3.1: Rozdíl mezi scénou se stíny a bez stínů. **Vlevo:** scéna bez stínů. **Vpravo:** scéna se stíny.

Shadow mapping [10, kapitola 12. *Stíny*] je metoda na tvoření tzv. tvrdých stínů, ty vznikají z nekonečně malých zdrojů světla, naopak měkké stíny vznikají z plošných zdrojů světla. Shadow mapping může vyvolat dojem měkkých stínů, jen pokud je použita vhodná filtrovací metoda. Shadow mapping vychází ze Z-bufferu, algoritmu pro řešení viditelnosti.

3.1.1 Princip

Scéna se vykresluje dvakrát. V prvním průchodu je renderována z pozice světelného zdroje, kde si ukládá pouze hodnotu hloubky do textury, tato textura se nazývá hloubková mapa *(depth map)*. Ve druhém průchodu je již scéna renderována z pozice kamery a každý bod je transformován do prostoru světla *(light coordinate system)*. Při rasterizaci se musí poslat do shaderu hloubková mapa a pro každý fragment se porovná hloubka. Tím se určí, zda je

fragment ve stínu či nikoliv. Jak je popsáno v knize Moderní počítačová grafika, metoda využívá algoritmus 1.

Algoritmus 1: Algoritmus Shadow Mapping

1. Zobrazení scény z pohledu světla L a uložení hodnoty ze Z-bufferu do hloubkové mapy H.
2. Zobrazení scény z pohledu kamery.
3. Převádění bodu z $[u, v, w]$ soustavy do prostoru světla L pro získání nové souřadnice $[x, y, z]$.
4. Získání hodnoty pixelu z hloubkové mapy $A = H[x, y]$.
5. Porovnání $A < z$, pokud je hloubka z mapy menší než skutečná, je pixel ve stínu.

Metoda funguje pouze pro jeden bodový směrový světelný zdroj. Pro všesměrový světelný zdroj existuje modifikace, kde místo jedné hloubkové mapy se vytvoří šest map, tím se světelný zdroj obalí do pomyslné krychle. Nevýhodou je, že scéna se překresluje šestkrát. Pokud je více zdrojů světla, pro každé světlo se vytvoří vlastní hloubková mapa.

3.1.2 Nežádoucí artefakty v shadow mappingu

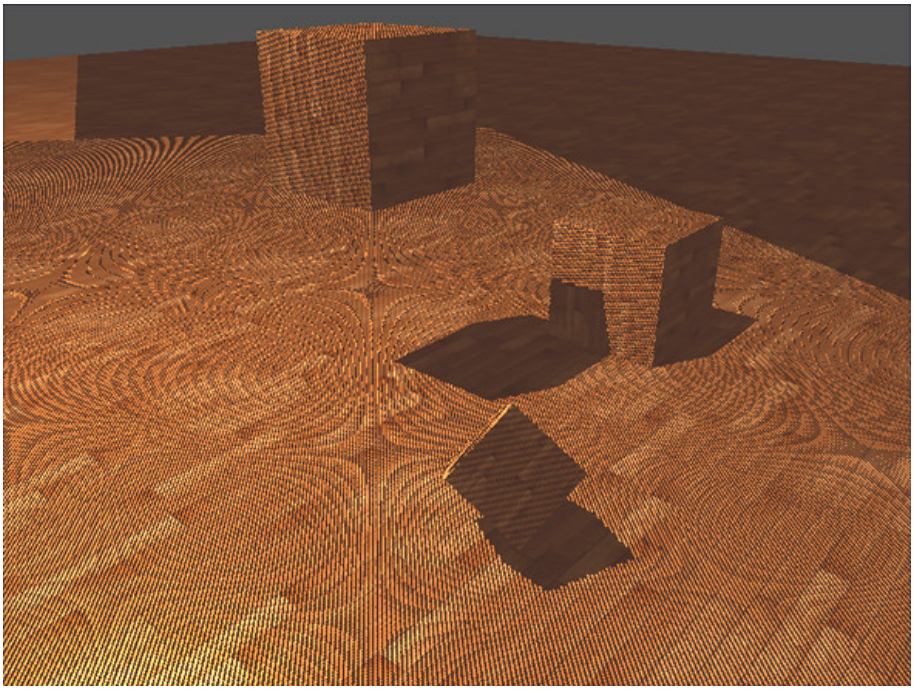
Metoda shadow mapping obsahuje spoustu nežádoucích grafických artefaktů. Tato práce se zabývá třemi z nich, a to aliasing, shadow acne a peter panning.

Jelikož tato metoda pracuje s rastrem dochází ke vzniku aliasingu. Pro odstranění aliasingu se používají například algoritmy PCF (percentage-closer filtering). V demu, kterým se práce zabývá, se používá matice 3×3, vybere se okolí pixelu z hloubkové mapy a zprůměruje se barva s okolními pixely. Tímto dojde k rozmazání hran stínů, částečně se odstraní alias a vznikne dojem měkkých stínů. Dobrý popis PCF metod a ukázky implementací pro grafickou kartu jsou ve zdroji [2].

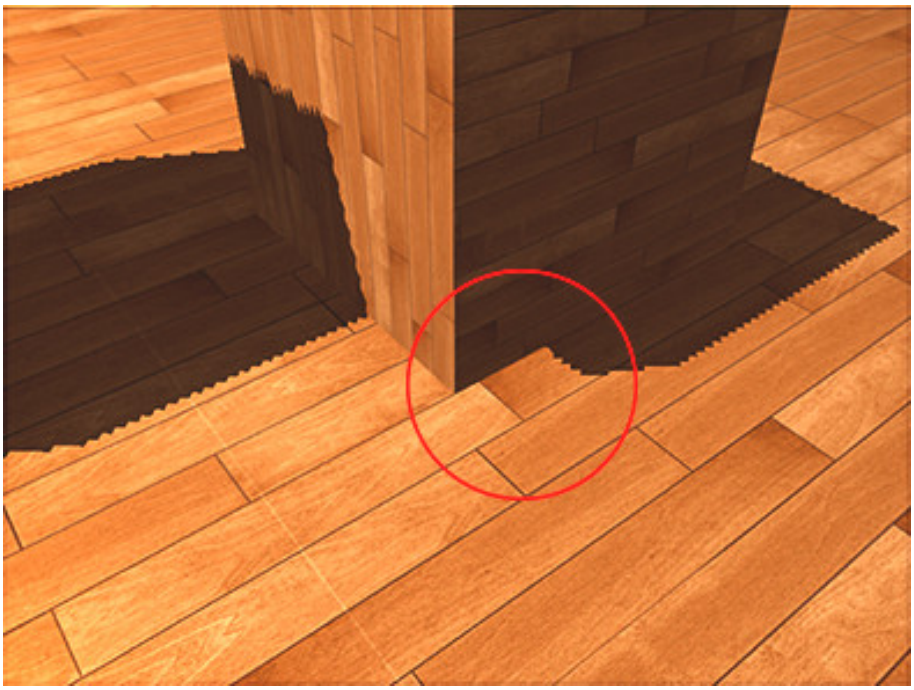
Dalším artefaktem je shadow acne popsaný Joey de Vriesem [9] v kapitole zabývající se shadow mappingem. Jedná se o to, že hloubka polygonů je nesprávně vyhodnocena a tyto polygony vrhají stín samy na sebe. Tento jev vzniká v důsledku omezeného rozlišení hloubkové mapy. Více fragmentových vzorků má stejnou hodnotu, pokud jsou relativně daleko od zdroje světla. Je-li vektor směru světla a normály objektu pod ostřejším úhlem, může několik fragmentů přistupovat k jednomu hloubkovému texelu, což má za důsledek, že někdy je detekován stín nad a jindy pod plochou objektu, tím vzniká pruhový vzor nesprávných stínů na ploše (obrázek 3.2).

Řešením je nastavení offsetu (zvaným bias) hloubky dané plochy, tím se posunou fragmenty a díky tomu nebudou nesprávně ležet pod povrchem objektu.

Pokud je nastaven velký offset, vede to k dalšímu artefaktu zvanému Peter Panning (obrázek 3.3). To znamená, že se vytvoří mezera mezi stínem a objektem. Možné řešení je opět snížit offset, to ale může vést zpátky k shadow acne, nebo zapnout kreslení všech stěn objektu, v OpenGL, sekce 4.1.1, pomocí funkce *„glCullFace“*.



Obrázek 3.2: Na podlaze i kostkách jdou vidět střídavé pruhy stínu, tento artefakt se nazývá Shadow acne. Převzato z [9].

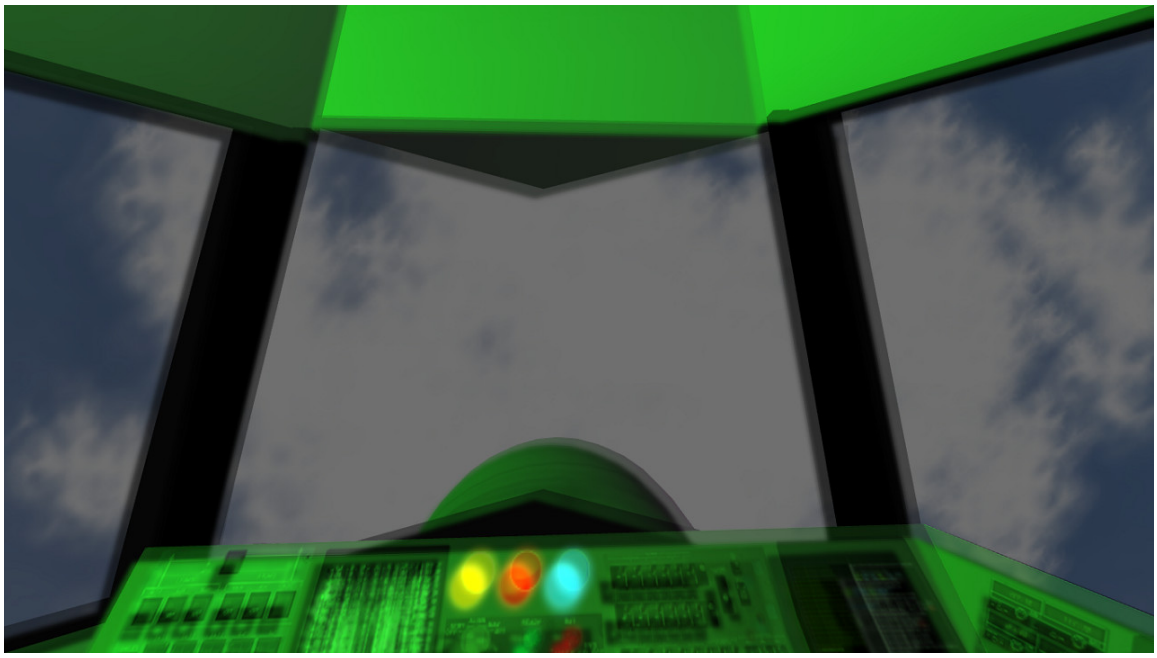


Obrázek 3.3: Na obrázku lze vidět v kroužku odskok stínu od objektu, tento artefakt se nazývá Peter Panning. Převzato z [9].

3.2 Rozmazání obrazu Gaussovou funkcí

Tato sekce je inspirována článkem od Daniela Rákose [6]. Rozmazání obrazu pomocí Gaussova rozložení je post-processingový efekt pro plošné rozmazání barev ve snímku. Technika vychází z Gaussovy funkce. Je to široce používaný efekt, typický pro redukování obrázkového šumu nebo redukování detailů. Používá se spíše v úpravě fotografií, v grafice zobrazované v reálném čase je většinou součástí jiných efektů (např. bloom, sekce 3.3). Rozmazání obrazu pomocí Gaussova rozložení redukuje vysoké frekvence obrázkových komponent, takže pracuje jako filtr dolní propusti. Matematická aplikace Gaussova rozmazání je totožná s konvolucí obrázku Gaussovou funkcí. Využívá se hlavně pro zlepšení vizuálního efektu při pohybu objektů a kamery v animaci.

Výpočet jádra Gaussovy funkce pro tuhle práci jsem provedl pomocí online kalkulátoru (http://dev.theomader.com/gaussian-kernel-calculator/).



Obrázek 3.4: Snímek rozmazaný pomocí Gaussova rozložení ve scéně v kombinaci s ostrým snímkem.

3.2.1 Gaussovo rozložení

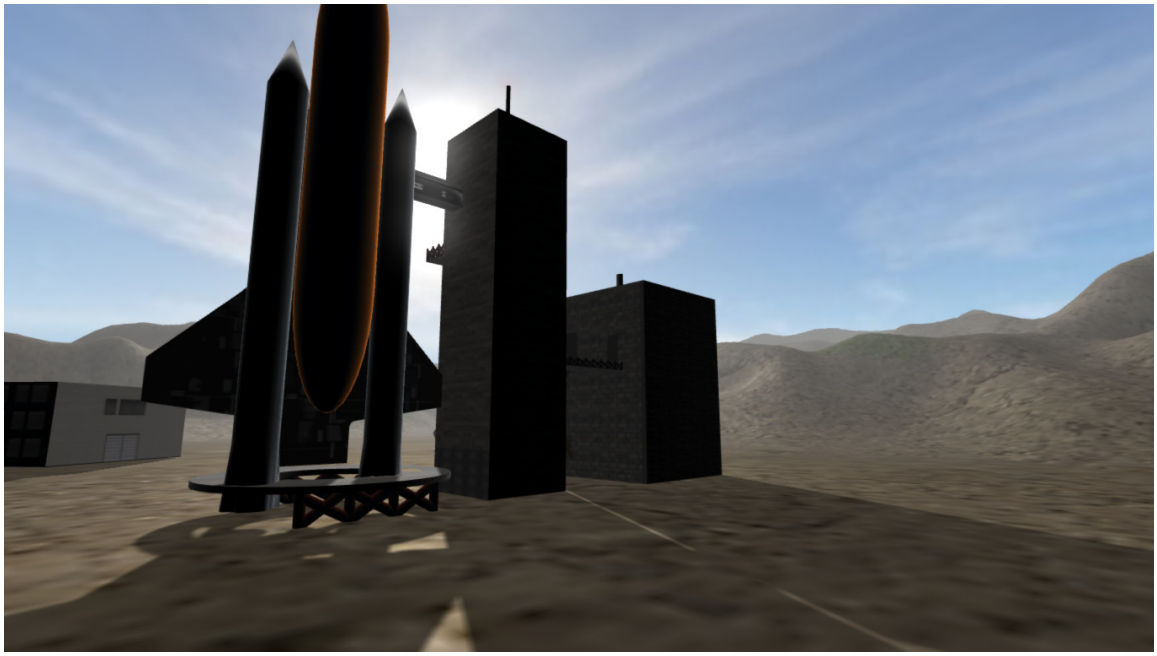
Někdy též normální, patří mezi nejdůležitější rozdělení pravděpodobnosti náhodné veličiny. Pomocí tohoto rozdělení je možné modelovat náhodné jevy z reálného světa. Gaussovo rozložení je popsáno funkcí:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.1}$$

Je to funkce o dvou parametrech, střední hodnoty μ a rozptylu σ^2. Normované Gaussovo rozložení se získá dosazením do Gaussovy funkce za $\mu = 0$ a $\sigma^2 = 1$. Tato sekce byla inspirována knihou od J. de Vriese [9] v kapitole Pokročilé osvětlení zabývající se efektem bloom v sekci o Guassovu rozložení.

3.3 Bloom

Jedná se o post-processingový efekt, který dodává světelným zdrojům ve scéně vizuální dojem záření a ohyb světla přes hrany objektů (obrázek 3.5). Vychází z faktu, že v reálném světě nedokážou čočky dokonale zaostřit, a pak se zdá, že se světlo láme přes objekt.

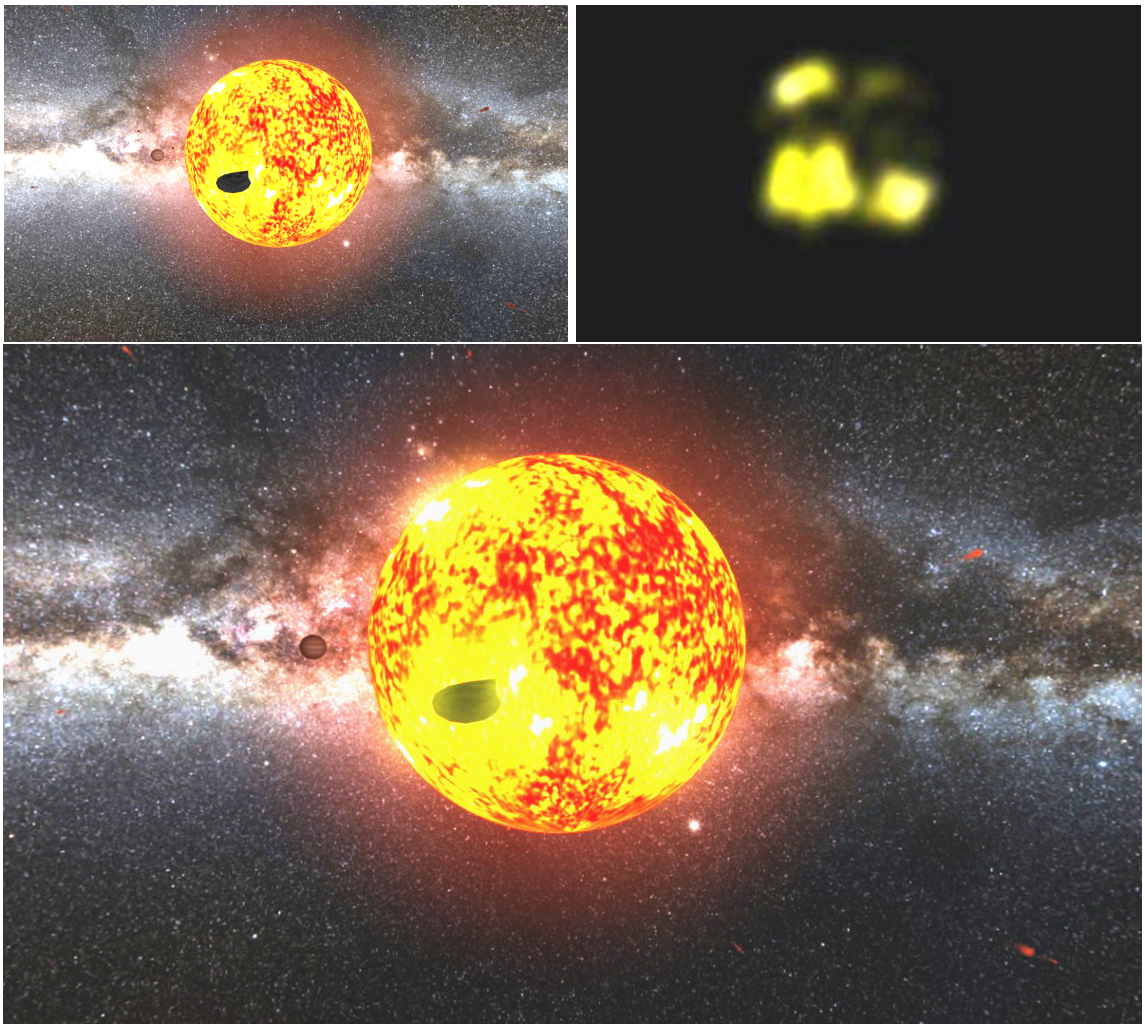


Obrázek 3.5: Světlo ze slunce lámající se přes most.

Efekt bloom ke své implementaci využívá rozmazávací filtr, v této práci bude využívat již zmíněné Gaussovo rozmazání (sekce 3.2), jehož výsledek se kombinuje s originálními barvami scény. Většina vizuální kvality je dána právě kvalitou a typem rozmazávacího filtru. Tato sekce vychází z kapitoly Pokročilé osvětlení v sekci zabývající se Bloom efektem ze zdroje [9].

3.3.1 Princip

Scéna se vykresluje běžným způsobem a její fragmenty jsou ukládány do dvou textur (tzv. *Multiple Render Targets*). Jedna z textur obsahuje originální barvu scény, druhá pouze barvy zářících objektů. Toho se docílí pomocí nastavení prahu intenzity fragmentu, pokud intenzita překoná práh, je uložena barva fragmentu do textury, pokud se tak nestane, je do textury uložena černá barva, tím se získá „prahová textura“. Prahová textura se podvzorkuje (*downsampling*), což zvýší rozsah záře a sníží se požadavky na výkon grafické karty. Na prahovou texturu se aplikuje Gaussovo rozmazání v horizontálním i vertikálním směru. Výsledná rozmazaná prahová textura je to, co se používá k získání zářícího efektu. Poté se barevné složky rozmazané prahové textury přičtou ke složkám textury, která obsahuje originální barvy scény. Vzhledem k tomu, že prahová textura je rozmazána v obou směrech, vzniká dojem, že jasné části scény vyzařují světlo. Výpočet bloom efektu není výkonově náročný a značně zlepšuje kvalitu a vzhled vykreslování v reálném čase. Ukázky implementace a podrobný popis je dobře zpracován ve zdroji [3].



Obrázek 3.6: Textura s originální barvou scény **vlevo nahoře**, prahová, podvzorkovaná a rozmazaná textura **vpravo nahoře**, **v dolní části** obrázku je spojení těchto textur do jedné (výsledná scéna).

3.4 Phongův osvětlovací model

3.4.1 Základní popis světla

Pomocí světla lze vizuálně vnímat svět a barvy. Putování světla prostorem je základem pro tvorbu virtuálních scén a jejich zobrazování. Jak je již zmíněno v knize Moderní počítačová grafika [10, kapitola 10. *Světlo*], světelné jevy nejsou triviální a jejich simulace je založena na komplikovaných fyzikálních jevech. Jelikož světlo má dualistickou povahu, lze ho popsat, jak vlnami, tak částicemi. Nauka o světle se dělí do následujících kategorií:

- **Geometrická optika** – světelné paprsky se dají popsat geometrickými pravidly
- **Vlnová optika** – modeluje světlo jako elektromagnetické vlny a popisuje jevy, kde geometrická optika selhává
- **Elektromagnetická optika** – zahrnuje vlnovou optiku a popisuje polarizaci světla
- **Fotonová (*kvantová*) optika** – vysvětluje interakci světla s materiálem

Počítačová grafika se hlavně zabývá světlem z pohledu geometrické optiky. Pro simulaci světla v počítačové grafice platí, že světlo se šíří přímočaře, rychlost je nekonečná a není ovlivněno gravitací nebo elektromagnetickým polem. Světlo lze popsat bodem umístění a směrovým vektorem.

Viditelná šířka pásma elektromagnetického spektra je 380-720nm, v tomhle rozsahu dokáže lidské oko rozlišit až 400 000 různých barev. Vnímání barev je důsledkem toho, že dopadne-li světlo na objekt, tak objekt určité frekvence pohltí, jiné odrazí a právě odražené frekvence jsou vnímány jako barva objektu. Barvy v počítačové grafice jsou nejčastěji reprezentovány modelem barev RGB (*red, green, blue*).

3.4.2 Popis metody

Phongův osvětlovací model je empirický (není založen na fyzikální podstatě) osvětlovací model, který se zabývá výpočtem odraženého světla. Skládá se ze tří světelných složek Ambient, Difuze a Specular (odraz). Výsledná barva objektu je dána součtem těchto složek (obrázek 3.10).

Složka Ambient

Popisuje odražené světlo ze všech směrů a nespecifikovaných zdrojů. Jedná se o okolní světlo, které má simulovat vlastnost rozptýlení světla. Je většinou bílé a pro celou scénu konstantní. Značí se I_a a je vyjádřeno vztahem:

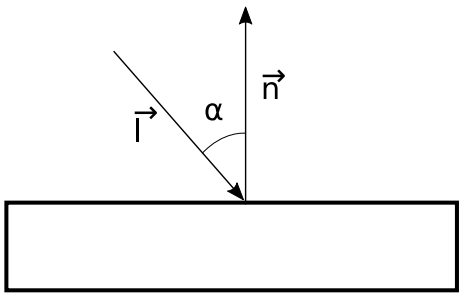
$$I_a = I_A r_a, \tag{3.2}$$

kde r_a značí koeficient pro schopnost materiálu odrážet okolní světlo.

Díky složce Ambient nejsou odvrácené strany objektů od světelných zdrojů černé. Při zvyšování počtu světelných zdrojů se složka Ambient snižuje, aby nedošlo k přesvícení scény.

Složka Difuze

Zastupuje celkové odražené světlo, značí se pomocí I_d. Jedná se o nejdůležitější vizuální část Phongova osvětlovacího modelu. Vychází z Lambertova zákona obrázek 3.7, který říká, že čím je směr dopadu blíž k normále, tím je intenzita světla (I) vyšší.



Obrázek 3.7: Na obrázku jsou znázorněny vektory dopadu světla a normály objektu pro lepší vysvětlení Lambertova zákona.

$$I_d = I \cos \alpha \tag{3.3}$$

Pokud skalární součin vektoru $\vec{l}$ a $\vec{n}$ je větší než 0, tak lze složku Difuze vyjádřit vztahem:

$$I_d = I_L r_d (\vec{l} \cdot \vec{n}) \tag{3.4}$$

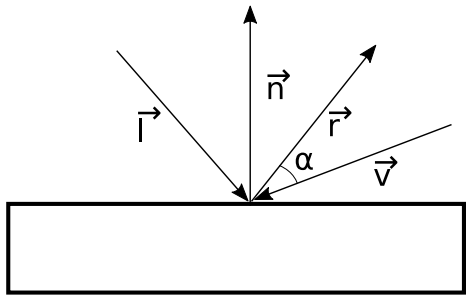
Ale je-li skalární součin vektoru $\vec{l}$ a $\vec{n}$ menší než 0, tak povrch objektu je odvrácen od zdroje světla a složka Difuze I_d je nulová. Koeficient odrazu r_d se většinou se shoduje s koeficientem odraženého světla r_a. Barevné složení paprsku světla je značeno jako I_L.

Složka Specular

Simuluje odlesk od povrchu a jeho barva je spíše barva světla než objektu. Lze jí popsat vztahem:

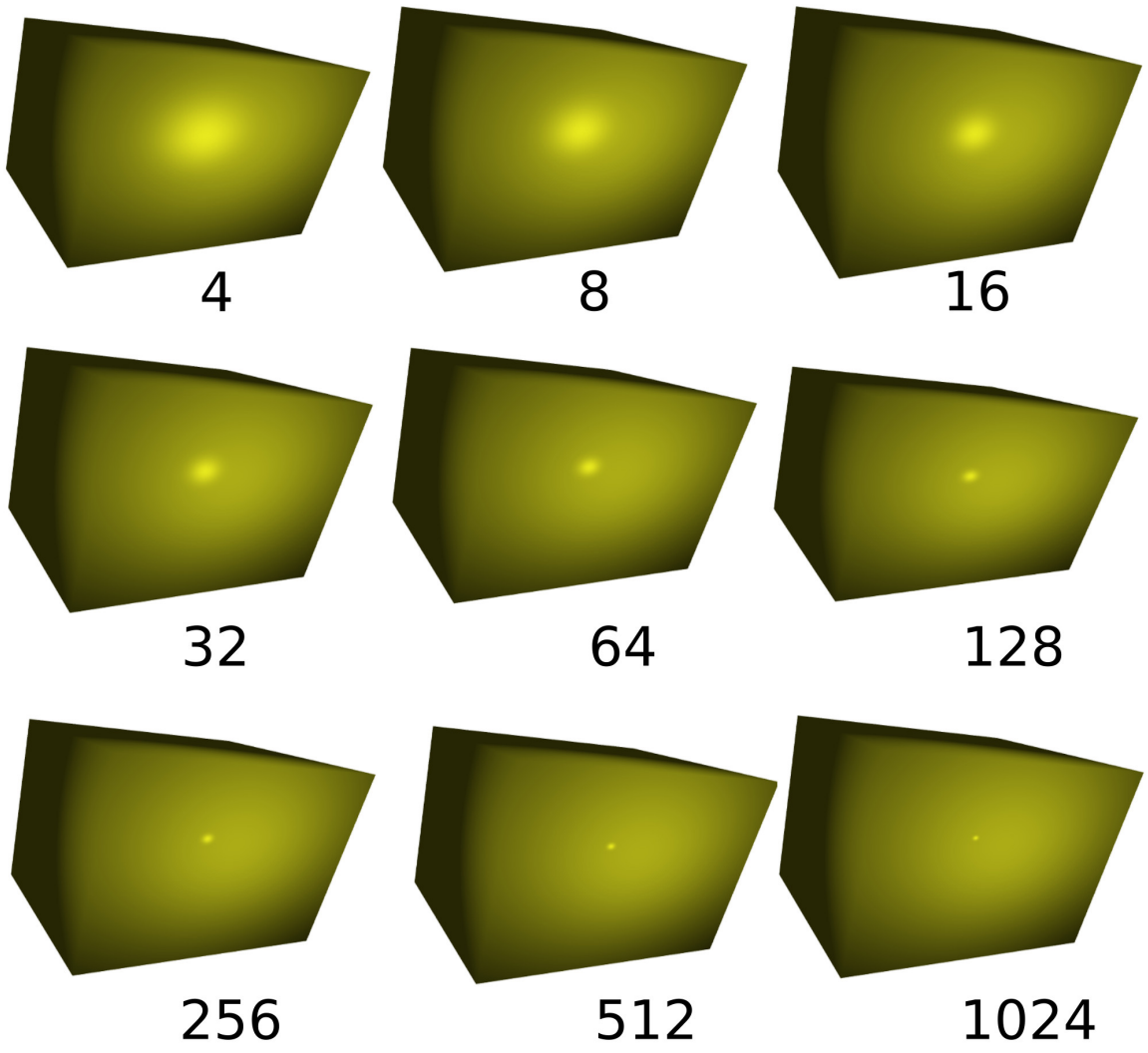
$$I_s = I_L r_s (\vec{v} \cdot \vec{r})^h, \tag{3.5}$$

kde $\vec{v}$ označuje normalizovaný vektor pohledu na objekt, vektor $\vec{r}$ zobrazuje zrcadlový odraz dopadajícího světla. Koeficient r_s určuje míru odražené Specular složky. Koeficient h je skalár a určuje ostrost odrazu. Nabývá hodnot v intervalu $< 1, \infty >$, přičemž $h = \infty$ by odpovídalo dokonalému zrcadlu, ovlivnění odrazu hodnotou h znázorňuje obrázek 3.9. Grafické znázornění složky Specular obrázek 3.8.

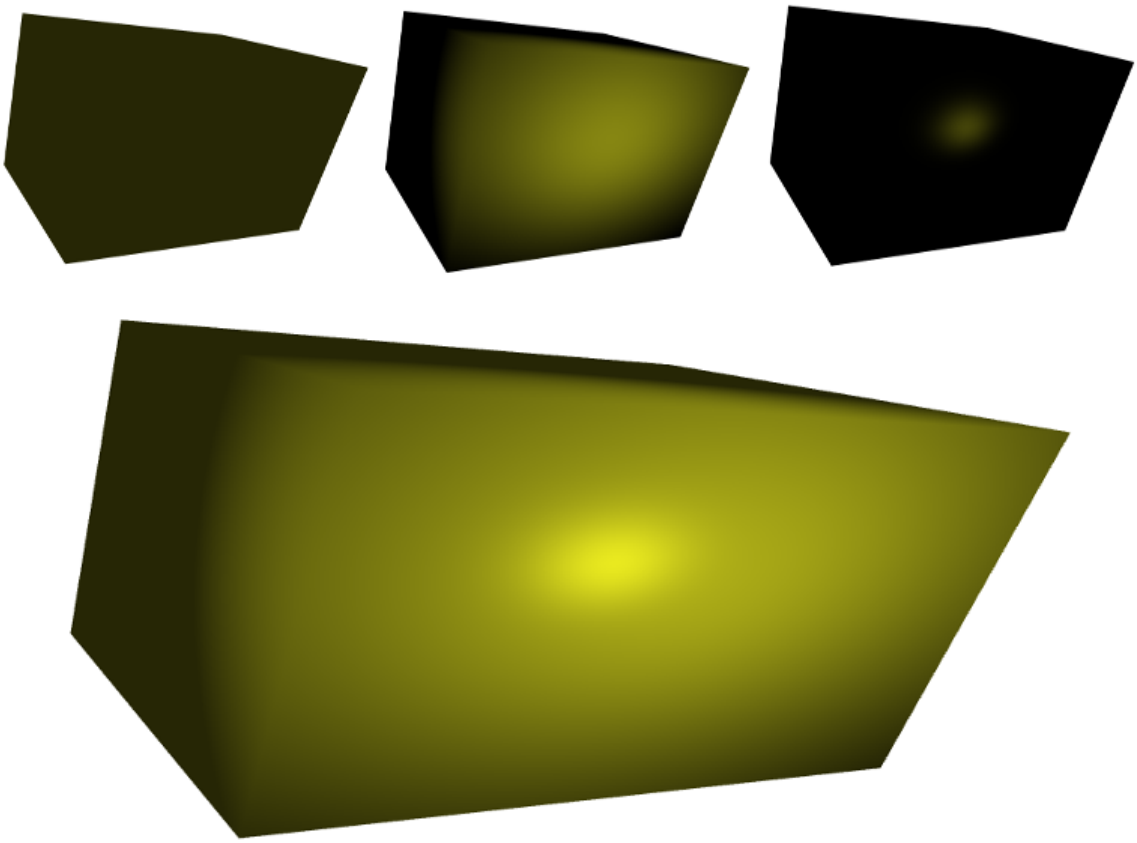


Obrázek 3.8: Grafické znázornění vysvětlující všechny vektory použité ve vzorci pro výpočet složky Specular.

Ačkoliv Phongův osvětlovací model není založen na fyzikální přesnosti, je v počítačové grafice široce používán. Díky jednoduché implementaci a efektivitě se stal standardem v aplikacích využívajících vykreslování v reálném čase a má podporu i v hardwaru.



Obrázek 3.9: Vizuální vliv koeficientu h. S narůstajícím h získává objekt iluzi lesklejšího materiálu.



Obrázek 3.10: Spodní obrázek ukazuje spojení všech složek Phongova osvětlovacího modelu, **vlevo nahoře** složka Ambinet, **uprostřed nahoře** složka Difuze, **vpravo nahoře** složka Specular.

3.5 Mlha

Jedná se o atmosferický jev, který v počítačové grafice může být použit z různých důvodů. V prvním případě může mlha zvyšovat úroveň realismu ve scéně. V dalším případě pomáhá uživateli určit, jak vzdálené jsou od něho objekty ve scéně. A jak je zmíněno v knize [4], efekt mlhy při správném použití skryje ořez objektu, který vykonává far plane. Pro výpočet mlhy platí vztah:

$$c_p = fc_s + (1 - f)c_f, \tag{3.6}$$

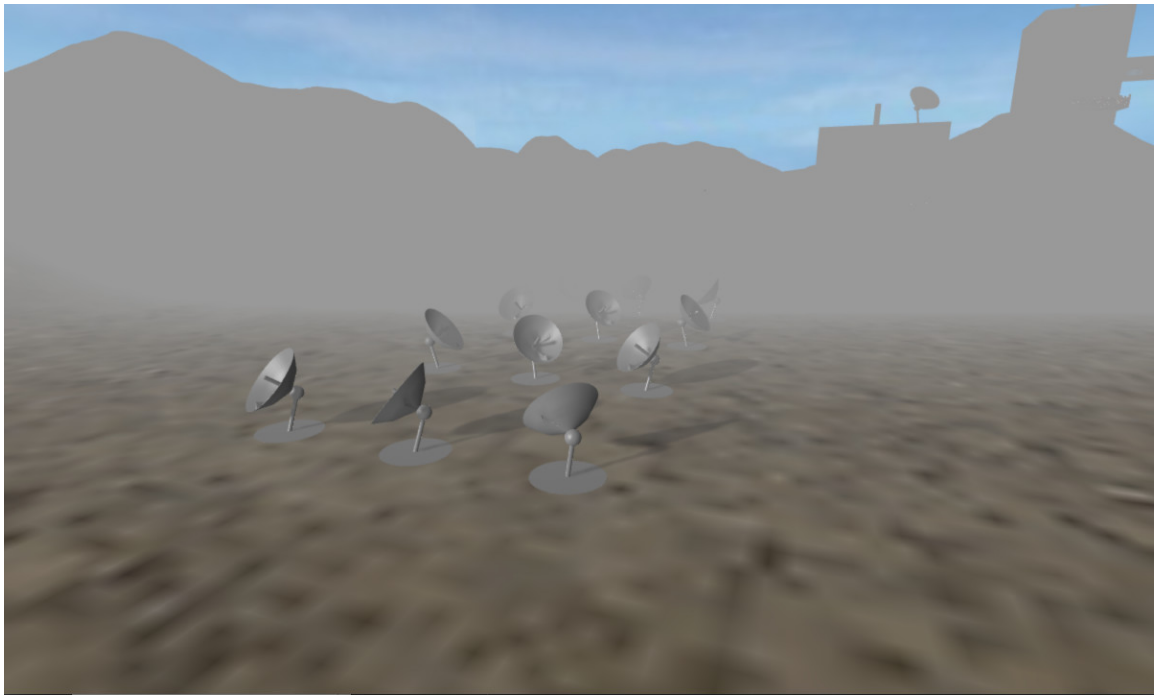
kde c_p je výsledná barva pixelu, c_s originální barva, c_f barva mlhy a f je mlhový faktor, který nabývá hodnot pouze v rozsahu $[0, 1]$.

3.5.1 Lineární mlha

Faktor mlhy je snižován lineárně a jsou jasně zadány skaláry pro začátek a konec mlhy (z_{start} a z_{end}). Výsledný faktor mlhy je počítán vztahem:

$$f = \frac{z_{end} - z_p}{z_{end} - z_{start}}, \tag{3.7}$$

kde z_p znázorňuje hodnotu hloubky pro pixel, kde je právě mlha počítána.

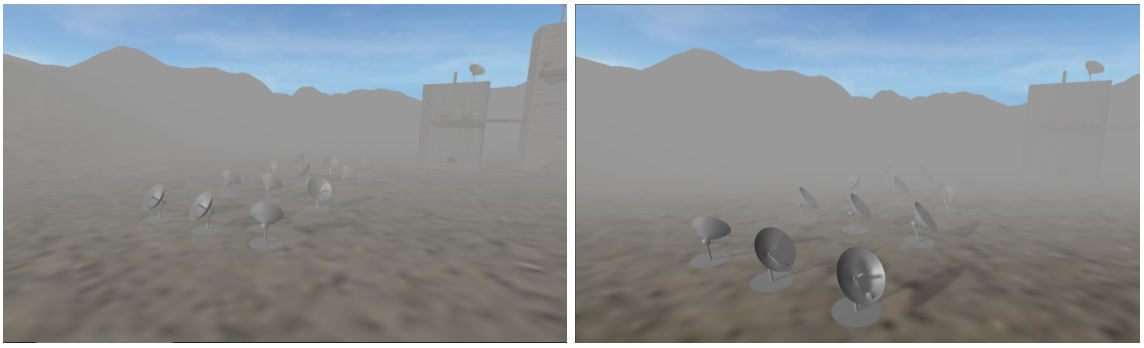


Obrázek 3.11: Lineární mlha, která nepůsobí realisticky, připomíná spíše zeď z mlhy.

3.5.2 Exponenciální mlha

Snižování faktoru mlhy probíhá exponenciálně, je použit nový skalární koeficient d_f, který značí hustotu mlhy. Výpočet faktoru mlhy je definován vztahem:

$$f = e^{-d_f z_p} \tag{3.8}$$



Obrázek 3.12: **Vlevo** exponenciální mlha, **vpravo** kvadratická exponenciální mlha, obě exponenciální mlhy používají stejný koeficient hustoty mlhy.

Kvadratická exponenciální mlha je definována vztahem:

$$f = e^{-(d_f z_p)^2} \tag{3.9}$$

Rozdíl mezi exponenciální a kvadratickou exponenciální mlhou je ukázán na obrázku 3.12. V demu, kterým se práce zabývá, je použita implementace kvadratické exponenciální mlhy ve fragment shaderu.

3.6 Tvorba terénu pomocí výškové mapy

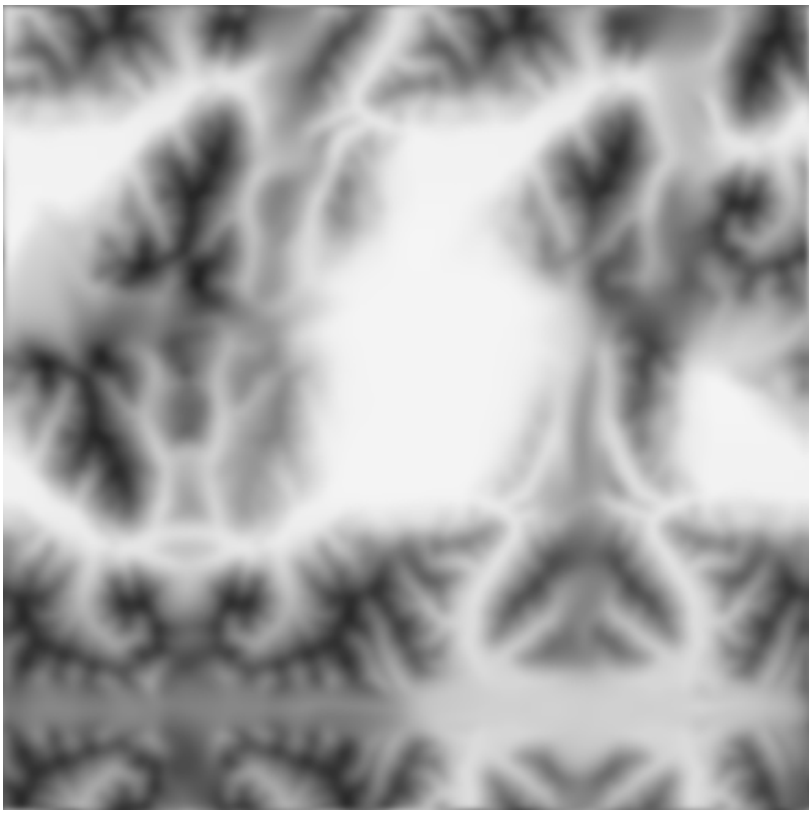
Tvorba terénu pomocí výškové mapy spadá do oblasti procedurálního generování, to znamená získávání grafických entit, jako jsou modely nebo textury, pomocí algoritmu. Základní komponentou této metody je výšková mapa (obrázek 3.13), tu znázorňuje textura obvykle ve stupních šedi, která v sobě nese informaci o výšce terénu. Rozlišení textury má vliv na počet primitiv, z kterých bude výsledný terén poskládán, ale nemělo by mít vliv na velikost terénu.

Tato metoda se nejčastěji spojuje s implementací Perlinova šumu pro procedurální generování textury, ale tato práce se implementací Perlinova šumu nezabývá. Výšková mapa v této práci byla vygenerována softwarem Gimp[1], kde byl do textury generován šum a následná manuální úprava rovných ploch uprostřed mapy.

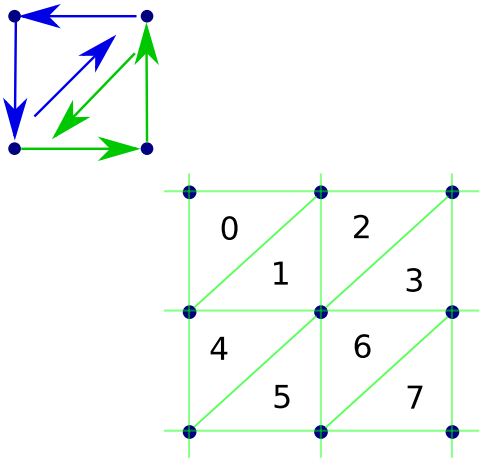
3.6.1 Princip

Algoritmus pomocí výškové mapy sestrojí čtvercovou síť z trojrozměrných bodů, souřadnici x určuje šířka textury, souřadnici z výška textury a souřadnici y hodnota na daném pixelu. Poté se čtvercová síť převádí na trojúhelníkovou síť, toho se docílí vytvořením index bufferu (ukládá pouze pořadí bodu k vykreslení). Trojúhelníky se musí zvolit ve správném směru, aby *face* směřoval do scény (obrázek 3.14).

[1] Oficiální stránky https://www.gimp.org/



Obrázek 3.13: Ukázka výškové mapy použité v demu. Obrázek má invertované barvy pro tisk, takže bílá je nejnižší bod a černá nejvyšší.



Obrázek 3.14: Triangulace čtvercové sítě. Obrázek znázorňuje správné zvolení směru (podle hodinových ručiček) trojúhelníků.

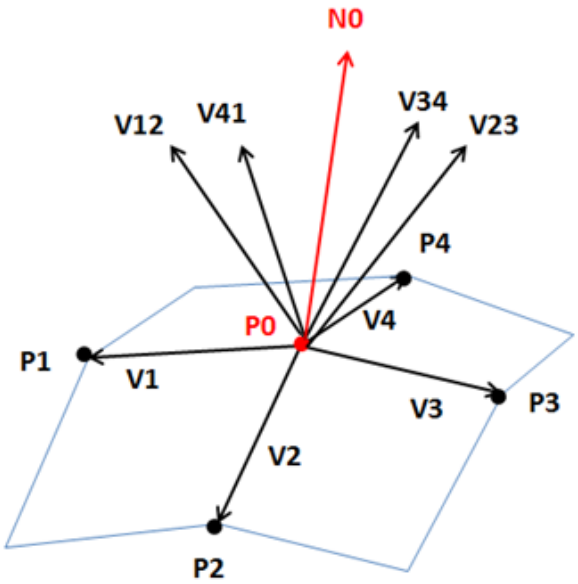
Určování texturovacích souřadnic pro objekt je popsáno vztahem:

$$u_x = \frac{x}{W-1}, \tag{3.10}$$

$$u_y = \frac{y}{H-1}, \tag{3.11}$$

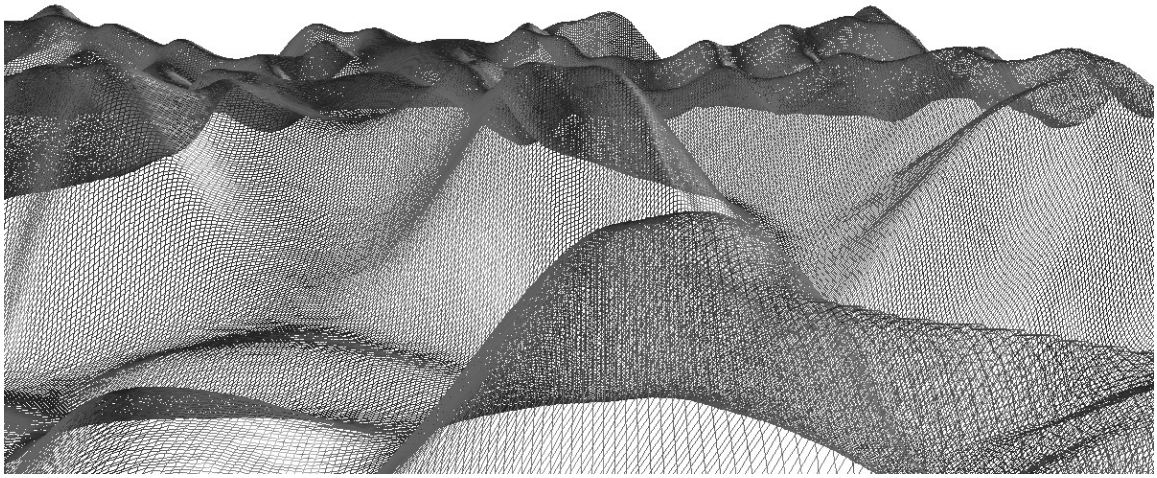
kde x značí aktuální sloupec ve výškové mapě, y aktuální řádek, W celkovou šířku mapy, H celkovou výšku mapy a $[u_x, u_y]$ výsledné texturovací souřadnice.

Dále se vypočítávají normály. Normály jsou potřeba pro korektní osvětlení objektu (viz sekce 3.4). Pro každý bod z čtvercové sítě se počítá s čtyřokolím bodů, pomocí kterých se vypočítají směrové vektory mezi aktuálním bodem a každým bodem z okolí. Z vypočtených vektorů se provede vektorový součin s každým vektorem, s kterým svírá úhel (obrázek 3.15), tím se získají dílčí normály mezi jednotlivými body. Pro výpočet výsledné normály se sečtou všechny dílčí normály a výsledek se normalizuje. Pokud aktuální bod z čtvercové sítě je krajní a nemá celé čtyřokolí, nahradí se chybějící bod aktuálním bodem, tím dostane dílčí normála nulovou hodnotu a neovlivňuje výpočet výsledné normály.

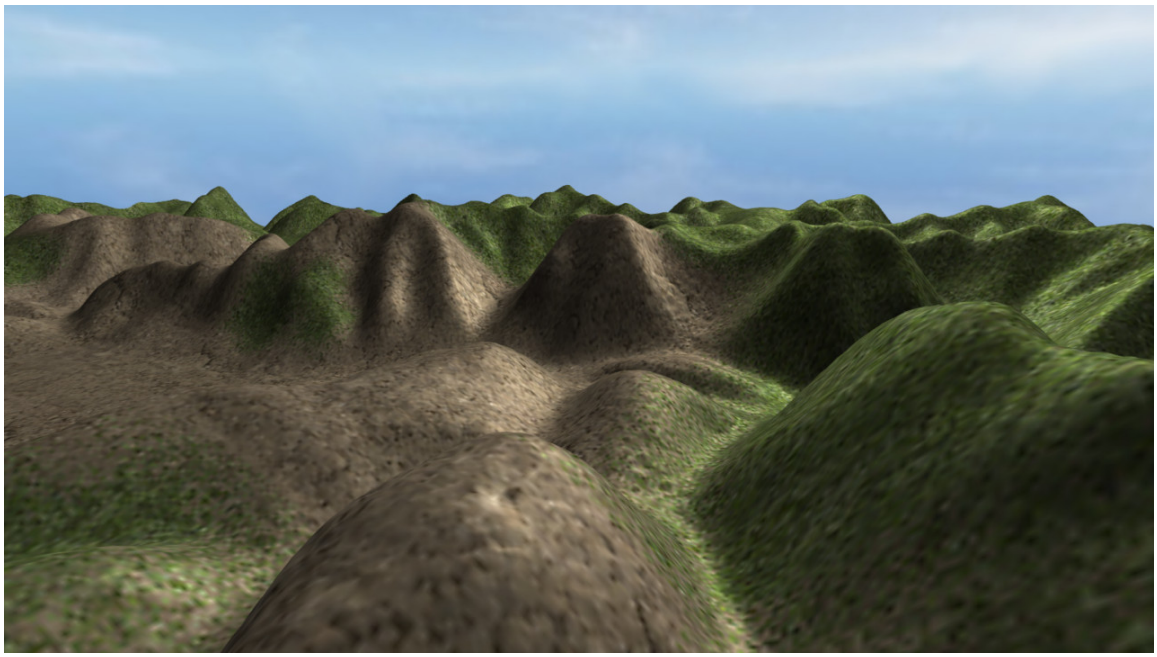


Obrázek 3.15: Znázornění bodů a normálových a pomocných vektorů. Převzato z [1].

Nevýhodou této metody je, že nepodporuje tvorbu skalních převisů nebo jeskyní. Některé informace byly použity ze zdroje [1].



Obrázek 3.16: Drátěný model výsledného terénu.



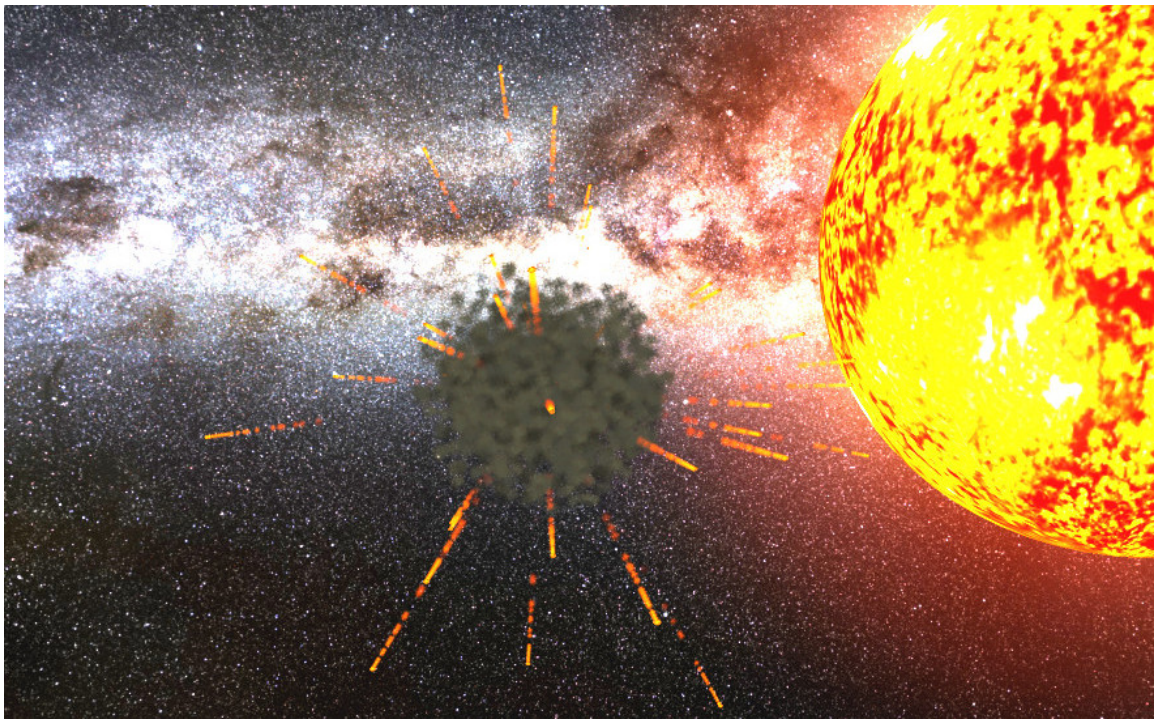
Obrázek 3.17: Výsledný otexturovaný terén.

3.7 Částicové systémy

Systémy částic se využívají pro modelování objektů s nejasným členitým tvarem, nebo pro dynamicky měnící se objekty, které není možné reprezentovat jako povrch (např. oheň, dým, exploze, déšť atd.). Částice jsou reprezentovány jako jednoduché malé objekty, které jsou od sebe odděleny. Částice jsou obvykle součástí animace, z toho plyne, že každá částice má své vlastnosti, jako je rychlost, směr, poloha, barva, tvar aj. (záleží, co částice simulují). Každá částice má i svůj život (*lifetime*). Podstatou je, že částice jsou řízeny po celý jejich život od vytvoření, jejich pohyb, změnu, až po odstranění. Částice, které překročily dobu svého života, zanikají a už se dále nevykreslují.

Částice jsou vytvářeny emitory částic, to mohou být buď konkrétní oblasti, nebo mohou vznikat jako potomci jiných částic. Částice většinou využívají náhodných čísel pro inicializaci jejich vlastností.

Částice se také často využívají pro fyzikální simulace, jak je uvedeno v knize Moderní počítačová grafika [10, kapitola 8. *Procedurální modelování*], kde se zmiňuje simulování tření u prototypů aut a letadel ve větrném tunelu pomocí částicových systémů.



Obrázek 3.18: Exploze pomocí částicového systému.

Obvykle jsou částice tvořeny alfa texturou, která jim udává tvar, a v mnoha případech jsou billboardovány (natočeny vždy ke kameře). Před vykreslením částicového systému je nutno částice seřadit podle vzdálenosti od kamery, aby funkce pro míchání barev fungovala korektně. Také pro korektní míchání barev lze u částic vypnout zapisování do hloubkového bufferu, ale v tomto případě se musí částice vykreslovat až nakonec scény po všech ostatních objektech.

Kapitola 4

Návrh a Implementace

Aplikace je implementována v objektově orientovaném návrhu programování, to znamená, že všechny metody jsou zapouzdřeny do tříd. Aplikace využívá dědičnosti a přepisování metod. Například všechny scény mají abstraktní bázovou třídu (chová se spíše jako rozhraní), z které ostatní třídy dědí a využívají přepisování virtuálních metod. Podobně je implementována i bázová třída pro grafické objekty, která řeší inicializaci a vykreslování těchto objektů.

Aplikace je psána jazykem C++ v ISO standardu 11. Využívá knihovnu OpenGL ve verzi 3.3 za použití jazyka GLSL (*OpenGL Shading Language*) pro programování grafického řetězce (*pipeline*) ve verzi 330 v core režimu. Jelikož komunikace s OpenGL API je přes funkce v jazyce C, jsou v aplikaci logicky odděleny části kódu a zapouzdřeny do tříd.

4.1 Použité knihovny

Všechny použité knihovny jsou primárně určené pro programovací jazyk C/C++, u některých z nich může existovat wrapper i na jiný programovací jazyk.

4.1.1 OpenGL API

OpenGL [7] je rozšířené multiplatformní API pro vysoce výkonové programování aplikací využívající 2D nebo 3D grafiku. Pomocí něho lze psát kódy přímo pro grafickou kartu (tzv. shadery) v jazyce GLSL. OpenGL je zaměřeno pouze na vykreslování, nemá podporu na vytváření kontextů a oken.

OpenGL se stále vyvíjí a vydávají se nové verze. Prvotní verze neměly programovatelný řetězec, ten přišel až s verzí OpenGL 2.0, to znamená, že jazyk GLSL byl dostupný až od této verze. Původně byl podporován pouze vertex a fragment shader. Vertex shader vypočítává a určuje pozici objektu, obvykle zde probíhá násobení maticí MVP (*model view projection* matice) s pozicí vrcholu, který je definován 4D vektorem, pro výpočet grafických transformací se používají homogenní souřadnice. Fragment shader nastavuje barvu objektu, přes tento shader mohou probíhat výpočty různých efektů. V core verzi 3.2 byl přidán geometry shader, který řídí zpracování primitiv, je volitelný a nemusí se použít. Verze core 4.0 přinesla tessallation shader, který rozděluje plochy vertexových dat na menší primitiva a vypočítává nová vertexová data, jako jsou poloha, barva, texturovací souřadnice. Aplikace, kterou se tato práce zabývá, využívá pouze vertex a fragment shader.

4.1.2 SOIL

SOIL[1] je multiplatformní knihovna pro načítání a ukládání obrázků, která dokáže přímo tvořit OpenGL textury. Má podporu na tvoření screenshotů. Podporuje většinu běžných formátů a bitových hloubek. Poslední verze od původního autora byla vydaná v roce 2008. Vývoj knihovny pokračuje dál na serveru github. Pro vývoj tohoto dema byla ideální volbou, protože dokáže načítat obrázky formátu png, a to jak s bitovou hloubkou RGBA, RGB, tak i grayscale (stupně šedi) a přitom není nadbytečně rozsáhlá.

4.1.3 SFML

SFML (*Simple and Fast Multimedia Library*)[2] je multimediální knihovna, která má podporu pro zobrazování oken a vykreslování 2D grafiky, pomocí nastavení kontextu s OpenGL API lze vykreslovat 3D grafiku, dále zpracovává uživatelské vstupy, podporuje paralelní programování (vlákna), audio a síťové programování. Nicméně, i když se jedná o tak rozsáhlou knihovnu, tato práce používá pouze audio část pro přehrávání zvuků a hudby v demu.

4.1.4 GLFW

GLFW[3] je multiplatformní open-source knihovna zaměřena právě na práci s 3D grafikou. Dovoluje spravovat a zobrazovat okna a odchytávat uživatelské vstupy. Lze vytvořit kontext pro OpenGL, OpenGL ES, nebo Vulkan API. Knihovna je stále aktivně podporována. Aplikace využívá tuto knihovnu ve verzi 3.2.1 pro zobrazování oken, tvoření OpenGL kontextu a pro uživatelské vstupy.

4.1.5 Assimp

Assimp[4] je multiplatformní knihovnou pro načítání 3D modelů. Knihovna podporuje spoustu formátů pro 3D modely, tato práce pracuje s formátem OBJ. Má podporu automatického převodu modelu objektu sestrojeného z jiných primitiv než jsou trojúhelníky právě na trojúhelníkové primitiva, také podporuje nastavení indexů do index bufferu a načítání skeletal animací aj. Je stále podporována a dostupná je pod BSD licencí.

4.2 Implementační detaily

Tato sekce se zabývá detailněji implementací popsaných grafických technik v kapitole 3, které se mírně liší od obecného popisu techniky. Dále popisuje řešení aliasingu v aplikaci a implementaci animace kamery.

4.2.1 Shadow mapping

Shadow mapping v této práci implementuje projekci světelného zdroje jako staticky nastavenou. To z důvodu, že scéna je také statická a není v ní žádný pohyb těles. Obvykle se projekce světelného zdroje přepočítává pro každou aktuální polohu kamery. Ořeže se box

[1] https://www.lonesock.net/soil.html.
[2] https://www.sfml-dev.org/.
[3] http://www.glfw.org/.
[4] http://assimp.org/.

kolem pohledového prostoru (*viewspace*) scény, podle množinového vztahu:

$$\big((S \cap V) \cup L\big) \cap S, \tag{4.1}$$

kde S znázorňuje scénu, V pohledový prostor a L pohledový prostor světelného zdroje.

Táto práce ještě obsahuje modifikaci ve výpočtu stínového faktoru, pokud se vykresluje odvrácená strana objektu od světelného zdroje, není počítán v tomto fragmentu stínový faktor. Zda se část objektu nachází na odvrácené straně od světelného zdroje, je ověřováno skalárním součinem vektoru normály objektu a vektoru směru dopadajícího světla (stejně jako u výpočtu intenzity světelné složky Difuze).

4.2.2 Phongův osvětlovací model

Phongův osvětlovací model v první scéně vychází z obecné implementace této techniky popsané v sekci 3.4. Nicméně ve druhé a ve třetí scéně je modifikován, a to absencí spekulární složky světla. Ve druhé scéně je vynechána z důvodu, že scéna neobsahuje žádné lesklé objekty. Ve třetí, vesmírné, scéně odebírala spekulární složka na reálnosti, protože v takovém měřítku se neodráží lesklé světlo tímto způsobem. Třetí scéna má rozšířený model ještě o simulaci šíření světla do dálky. S větší vzdáleností objektů od světelného zdroje intenzita světla klesá. Výpočet celkové barvy se počítá ve fragment shaderu pomocí vztahu:

$$c_{out} = c_{obj} \cdot \frac{I}{lk}, \tag{4.2}$$

kde c_{out} je výsledná barva fragmentu, c_{obj} je barva objektu, I je intenzita světla, l vzdálenost objektu od světelného zdroje a k je konstanta, která určuje, po jakých krocích bude světlo ztrácet svou intenzitu. Vztah by bylo možné rozšířit o výpočet konstanty k v závislosti na maximálním dosvitu světla.

4.2.3 Částicový systém

Pomocí systému částic jsou v aplikaci implementovány erupce, oblaka a exploze. Částice mají základní bod, od kterého se odvíjí tzv. emitor částic. Částice jsou zabaleny do třídy, která se podle vnějšího rozhraní chová jako jednoduchý grafický objekt.

Popis jednotlivých druhů částic

Erupce nevyužívají alfa texturu, pohybují se po kruhu, každá částice má přiřazen náhodný rádius a také rychlost, aby vytvořily dojem podlouhlého půlkruhu. Erupce jsou tvořeny na náhodných místech na povrchu Slunce, náhodnost zaručuje standardní funkce pro generování pseudonáhodných čísel *rand*. Tvoří se šest erupcí ve stejný čas, jakmile jedna z erupcí skončí, všechny její částice se nastaví jako neaktivní, začne se tvořit nová erupce na jiném náhodně vygenerovaném místě.

Oblaka mají v demu předem definované pozice, jejich základní body (emitory) se pohybují směrem k raketě, což vyvolává dojem, že raketa stoupá do vesmíru, takto zvolená implementace je z důvodu, že interiér rakety je tvořen více objekty, které by se musely pohybovat s kostrou rakety. Oblaka využívají alfa texturu, z toho důvodu jsou řazena podle jejich základních bodů pomocí *bubble sortu*. Jednotlivé částice nevyužívají řazení, pro lepší výkonost se částice nezapisují do hloubkového bufferu a díky tomu se míchání barev obejde bez nutnosti kreslit částice v daném pořadí. Tyto částice mají jednoduchou implementaci detekce kolizí. V momentě, kdy se počítá nová pozice každé částice, dojde k porovnání

pozice s obdélníkovou schránkou rakety, kterým se zjistí, zda částice leží uvnitř nebo vně rakety, pokud částice leží uvnitř, je nastaven příznak „kolize“ a částice není vykreslena.

Exploze je nejsložitější částicový systém tohoto dema, skládá se z více druhů částic (záblesk, prach, kousky horniny, kameny a oheň letící za kameny). Stejně jako u oblak částice nejsou řazeny, takže korektní míchání barev se zajišťuje nezapisováním hodnoty do hloubkového bufferu. Částice jsou rozděleny do druhů, protože každý druh využívá jiné textury, některé mají odlišný pohyb a rychlost. V prvním řešení tohoto částicového systému měly druhy částic i rozdílné shader programy, ale přepínání programů nebylo efektivní, proto jsou částice počítány v jediném shaderu, který využívá *multitexturingu*, tuto metodu by bylo možné také optimalizovat, a to použitím atlasu textur[5], protože přepínání jednotlivých textur v shaderu je náročné na výkon. Letící kameny jsou tvořeny objektem meteoru a jako jediné částice v demu nejsou billboardované a zároveň slouží jako emitory pro částice znázorňující letící oheň za kameny. Ostatní částice mají jako emitor základní bod, z kterého vychází. Směr částic je náhodný a je vypočítáván pomocí sférické soustavy souřadnic a následné transformace, vztah 4.3, do kartézské soustavy souřadnic, aby se dosáhlo kulového tvaru.

$$x = r \sin(\theta) \cos(\varphi), \tag{4.3}$$
$$y = r \sin(\theta) \sin(\varphi), \tag{4.4}$$
$$z = r \cos(\theta), \tag{4.5}$$

pro θ platí $0 \leq \theta \leq \pi$, pro φ platí $0 \leq \varphi < 2\pi$, r značí rádius.

4.2.4 Implementace audia

Implementace audia je zapouzdřena ve třídě *Audio_manager*, která je implementována podle návrhového vzoru Jedináček (existuje pouze jedna instance třídy). Třída řeší tvoření zvuků, hudby, přehrávání, nastavení souřadnic pozice zvuku a posluchače. Zvuky a hudba jsou uloženy odděleně a jsou ukládány pomocí kontejneru *std::map*. Přístup k jednotlivým prvkům je umožněn pomocí klíčového řetězce, který je většinou reprezentován názvem souboru. Lze spustit více zvuků paralelně. Logiku spouštění a správy zvuků řeší knihovna SFML, sekce 4.1.3.

4.3 Antialiasing

Metody antialiasingu jsou potřeba pro zmenšování nebo odstraňování aliasu. Alias je rušivý artefakt, který vzniká při rasterizaci obrazu. Nejčastějšími projevy jsou „zubaté“ hrany a poruchy textur. Kniha Moderní počítačová grafika [10, kapitola 2. *Obraz a jeho reprezentace*] definuje vznik aliasu tak, že vzorkovací frekvence při rekonstrukci signálu má hodnotu pod Nyquistovým limitem. Pokud má vzorkovaný signál frekvenci f_{max}, vzorkuje se tento signál frekvencí alespoň $2f_{max}$. Tím vzniká nová nízkofrekvenční informace, která nebyla přítomna v původním vzorkovaném signálu. Alias může také vzniknout, pokud se vzorkuje příliš pravidelně nebo příliš přesně.

Možnosti pro odstranění aliasu jsou zvětšení vzorkovací frekvence, v počítačové grafice je možné provést pomocí zvětšení rozlišení výstupních dat, z čehož plyne větší datová

[5] Odkaz na fórum zabývající se rozdílem mezi atlasem textur a spritesheetem https://gamedev.stackexchange.com/questions/69895/what-is-the-difference-between-a-sprite-sheet-and-a-texture-atlas (viděno 3.5.2018)

náročnost výstupního obrazu. Přefiltrováním výstupních dat (vyhlazením výstupních dat na konci celého procesu „post-processing“) u těchto metod dochází ke ztrátě detailů výstupních dat. Před-filtrováním vstupních dat se také zvyšuje vzorkovací frekvence, z toho plyne, že je k dispozici více vstupních vzorků na jeden výstupní, ale zachovává se výstupní rozlišení, u těchto metod dochází ke ztrátě výkonu. Aplikace implementuje metody supersampling a multisampling, které jsou implementovány v rámci třídy *Antialiasing*, tyto hodnoty jsou konfigurovatelné ze souboru *universe.config*, jeho podrobnější popis lze najít v příloze B. Dále implementuje antialiasing textur pomocí MIP mapování textur, tento druh antialiasingu OpenGL hardwarově podporuje, tudíž stačí jen z textury pomocí „glGenerateMipmap“ vygenerovat MIP mapu.

4.3.1 Supersampling

Supersampling patří do kategorie před-filtrování vstupních dat. Každý pixel je rozdělen na několik vzorků (tzv. subpixely), složení vzorků zajišťuje konvoluční filtr a vyhlazuje výsledný obraz (hrany i textury). Nevýhodou této metody je výrazný pokles výkonu. V aplikaci je supersampling implementován pomocí framebufferu, který nejprve inicializuje svoji velikost, ta je nastavována pomocí vstupního parametru, který udává násobek zvětšení od výstupního rozlišení. Poté jsou do framebufferu vykresleny objekty dané scény a framebuffer je předán výstupnímu framebufferu nebo pro post-processingové zpracování obrazu.

4.3.2 Multisampling

Multisampling patří také do kategorie před-filtrování vstupních dat. Oproti supersamplingu má hustější vzorkování pouze v oblasti hran objektů, uvnitř nebo mimo objekty je vzorkování řidší. Výrazně vyhlazuje hrany objektů, ale nevyhlazuje textury. Metoda je úspornější na výkon aplikace. V OpenGL má podporu v hardwaru. Multisampling je implementován pomocí speciálního framebufferu, který vykresluje data do multisample textury, tento framebuffer je poté předán pro zpracování supersampling framebufferem a dále předávání pokračuje jako v sekci 4.3.1 zabývající se supersamplingem.

4.4 Animace

Animací může být chápána jakákoliv dynamika objektů ve scéně v oblasti grafiky zobrazované v reálném čase. Animace může být buď deterministicky určena, a nebo vypočítávána pomocí algoritmů simulujících fyzikální jevy, kde stačí zadat počáteční podmínky. Podle knihy Moderní počítačová grafika [10, kapitola 18. *Počítačová animace*], jsou jedni z nejužitečnějších algoritmů algoritmy přímé a inverzní kinematiky, které se používají k simulaci pohybu vzájemně spojených objektových struktur. Počítačovou animaci lze rozdělit na nízkoúrovňovou a vysokoúrovňovou. Nízkoúrovňovou animací je myšlen popis pohybu objektu po spojité dráze. Vysokoúrovňová animace navíc může řešit kolize nebo implementovat již algoritmy fyzikální simulace. V aplikaci, kterou se tato práce zabývá, je implementována pouze nízkoúrovňová animace, a to jak objektů, tak i kamery.

4.4.1 Animace objektů

Objekty jsou animovány pouze v poslední vesmírné scéně tohoto dema. Jedná se o animaci těles pohybujících se po elipse a rotujících kolem své osy. Nejjednodušeji implementovaná je

animace pohybu asteroidů v pásu. Asteroidy se pohybují po kružnici konstantní rychlostí. Pohyb všech těles po elipse či kružnici je definován vztahem:

$$x = S_x + A\cos(\alpha)\, r, \tag{4.6}$$

$$y = S_y + B\sin(\alpha)\, r, \tag{4.7}$$

kde $[x, y]$ udávají výsledný bod, $S_{x,y}$ je střed elipsy, A je majoritní osa elipsy, B je minoritní osa elipsy a r je rádius. Pokud se určuje dráha po kruhu, odebere se ze vzorce majoritní a minoritní osa elipsy. Pohyb planet je počítán pomocí stejného vzorce, pouze vychází z prvního a druhého Keplerova zákona[6] (vycházelo se ze vzorců fyziky základní školy, zdroj [8]). Všechny údaje byly předpočítány a aplikace k nim přistupuje pomocí konstant v souboru *constants.h*.

4.4.2 Animace kamery

Animace kamery je rozdělena do tří na sobě nezávislých částí. První část řeší rozhlížení kamerou, druhá část pohyby kamery a třetí posílá hodnotu do framebufferu, který řeší ztmavení obrazovky pro střihy, tato hodnota je použita v aplikaci pro mixování originální barvy snímku s černou barvou. Jednotlivé definované animační pohyby jsou uloženy do front FIFO. Každý pohyb ve frontě si nese data o druhu pohybu a čas začátku a konce dílčí animace. Animační časy jsou normalizovány, nabývají hodnot $< 0; 1 >$. Pohyby pro rozhlížení jsou rozděleny na dva druhy. Prvním z nich je rozhlížení o daný úhel za určitý čas, tento pohyb využívá lineární interpolace, která je definována vztahem:

$$y = y_0 + (x - x_0)\frac{y_1 - y_0}{x_1 - x_0}, \tag{4.8}$$

kde v demu platí, že y je aktuální úhel pro snímek, y_0 počáteční úhel, y_1 koncový úhel, x znázorňuje aktuální čas animace, x_0 počáteční čas animace a x_1 koncový čas animace (nebere se v úvahu počáteční a koncový čas celé animace, ale pouze daného pohybu). Dalším druhem rozhlížejícího pohybu je sledování bodu, ten je vyřešen nastavením *LookAt* matice na daný bod.

Pohyby animační kamery jsou rozděleny na tři druhy, lineární pohyb po čáře, pohyb po kvadratické křivce a pohyb po kubické křivce. Lineární pohyb je implementován pomocí lineární interpolace bodu v prostoru, která je dána vztahem:

$$P = (1 - t)P_0 + t\, P_1, \tag{4.9}$$

kde P značí bod aktuální pozice, P_0 počáteční bod, P_1 koncový bod a t znázorňuje normalizovaný čas. Pohyby po křivkách jsou definovány Bézierovými křivkami.

Bézierovy křivky

Bézierovy křivky patří mezi aproximační křivky, což znamená, že křivka neprochází řídícími body. Vlastností Bézierových křivek je, že při změně jednoho řídícího bodu se změní tvar celé křivky, proto se tyto křivky dělí na segmenty (nejčastěji kubiky), které na sebe postupně navazují. Aplikace využívá Bézierovy kubiky, která je tvořena čtyřmi body, křivka vychází z prvního řídícího a končí v posledním. Tato kubika je definována vztahem:

$$P = (1 - t)^3\, P_0 + 3\,(1 - t)^2\, t\, P_1 + 3\,(1 - t)\, t^2\, P_2 + t^3\, P_3, \tag{4.10}$$

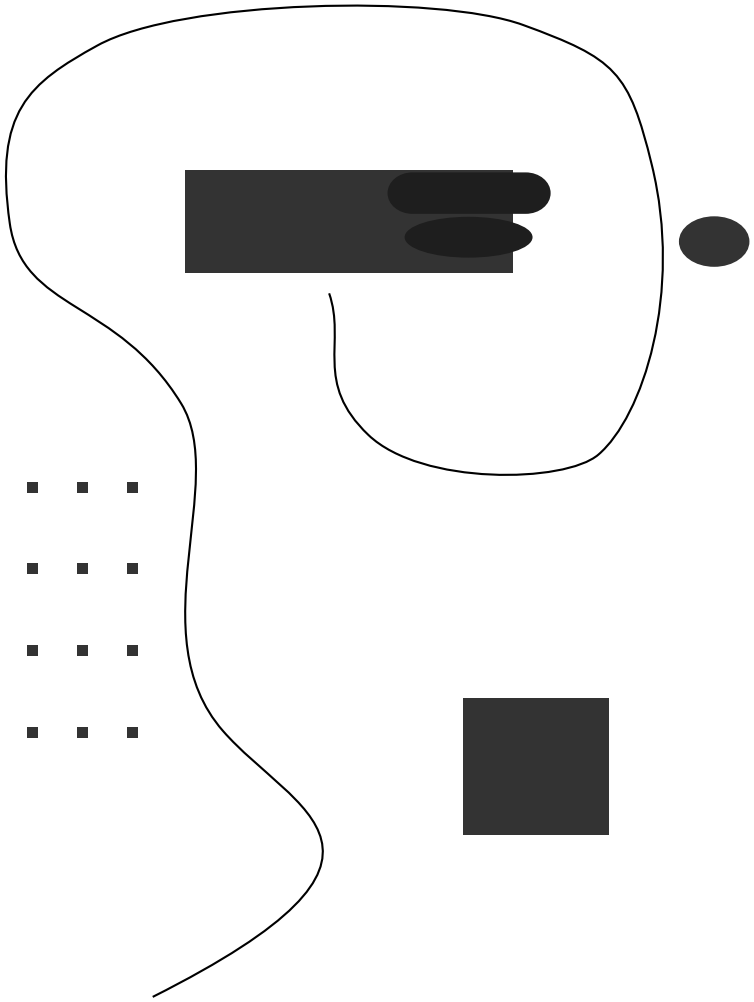
[6] Implementace tohoto fyzikálního jevu není pro demo důležitá, jev je použit pouze, aby poměry rychlostí a drah nebyly zcela zcestné.

kde P je aktuální bod, P_0 počáteční, P_3 koncový a P_1 a P_2 řídící body, t je normalizovaný čas. Dále aplikace podporuje využití kvadratické interpolace Bézierových křivek, která je definována vztahem:

$$P = (1-t)^2 P_0 + 2(1-t)\,t\,P_1 + t^2 P_2, \tag{4.11}$$

kde P znázorňuje aktuální pozici bodu, P_1 je řídící bod, P_2 koncový bod a t je normalizovaný čas.

V demu se nejprve definuje animace pro celou scénu, to je uskutečněno naplněním front a poté se spouští každým snímkem prvek, který je první ve frontě s ověřením, zda nastal jeho animační čas. Po uplynutí animačního času je prvek odebrán z fronty. Křivky pro scény byly navrženy pomocí aplikace Inkscape, oficiální stránky https://inkscape.org.



Obrázek 4.1: Křivkou je znázorněna dráha animace kamery, šedé kostky znázorňují objekty ve scéně.

Kapitola 5

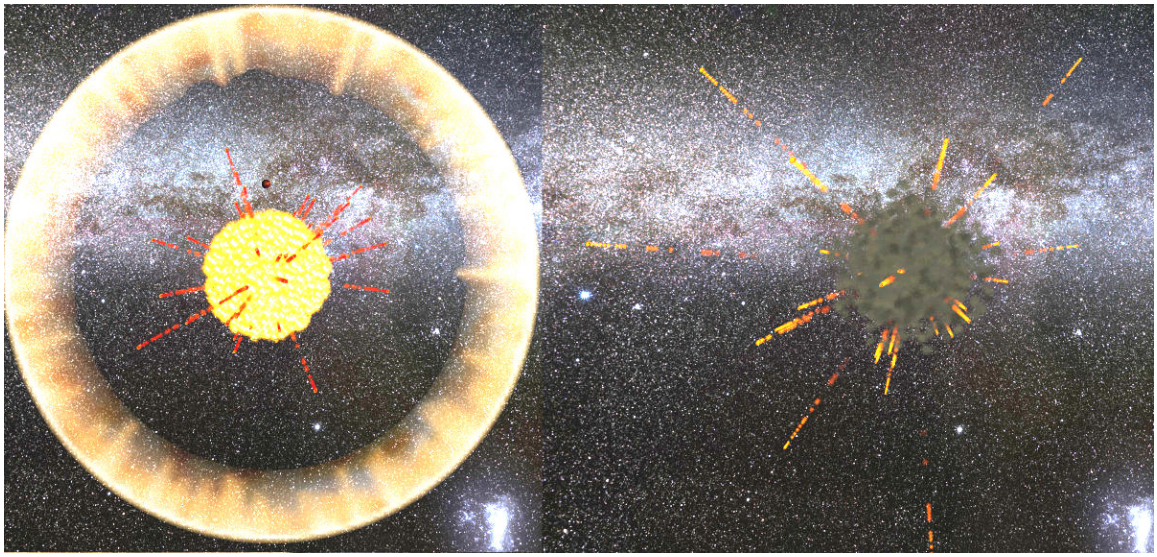
Testy a výsledky

Tato kapitola se zabývá testováním aplikace, a to jak vizuálním na uživatelích, tak i výkonovým. Výsledky výkonových testů jsou zobrazeny pomocí grafů. Obsahují popisy konkrétních závislostí vynesených do grafů a také zmiňují konkrétní hardware, kde byly testy prováděny. Výkonových testů bylo provedeno více než uživatelských.

5.1 Uživatelské testy

Testeři uživatelských testů byli lidé ve věkovém rozptylu 20 až 70 let. Nejprve byli testeři seznámeni s tématem, vzápětí jim bylo celé demo puštěno, poté byla zpětná vazba testerů zpracována v podobě dotazníku. V prvních testech měli testeři výhrady k malému množství grafického materiálu v demu, jelikož obsahovalo pouze vesmírnou část. Na základě těchto výhrad byla později doplněna scéna na Zemi a odletová scéna. V dalších testech testeři vydávali výbuch za nereálný kvůli použitým barvám, viz srovnání barev na obrázku 5.1.

Výsledky testování finální aplikace dopadly pro demo velmi dobře. Většina testerů, až na výjimky, hodnotila celkovou grafickou stránku s efekty jako velmi dobrou. Průměr výsledků této části je 4,04 (podle zvolené stupnice, kdy číslo 5 označuje nejpozitivnější hodnocení). Nejvíce uživatele zaujala vesmírná scéna a také zpracování erupcí na Slunci a exploze. Dále kladné hodnocení zahrnovalo i zvuk ve scéně na Zemi a ve vesmíru, naopak zvuk motoru rakety v odletové scéně uživatelům spíše narušoval dobrý pocit z animace. Záporně uživatelé hodnotili nízké rozlišení textur, které bylo do výsledného dema sníženo za účelem dosáhnutí menší paměťové náročnosti. Následně někteří uživatelé považovali grafiku za zastaralou, což také mohlo být příčinou menšího rozlišení textur, ale někteří naopak oponovali, že demo využívá moderní počítačovou grafiku, tento rozpor názorů mohl být způsoben tím, že většina testerů nebyla z oboru IT a počítačové grafiky. Uživatelé srovnávali demo spíše s vesmírnými dokumenty nebo moderními počítačovými hrami. Poté byla hodnocena plynulost animace. Ve většině případů uživatelé považovali animaci za plynulou a dobrou. Někteří si všimli občasných záseků animace, které jsou způsobeny tím, že výsledná animace se skládá z dílčích animací a výpočet úhlu v přechodu z jedné animace do druhé s sebou nese mírnou nepřesnost, která způsobuje právě trhnutí kamery v demu. Dále si často uživatelé stěžovali na dlouhou prodlevu mezi scénami, ale ta je způsobena dealokací objektů z předchozí scény a také alokací a inicializací nových objektů následující scény. Testování proběhlo formou dotazníku, který je přiložen v příloze D. Na CD, jehož obsah je dodán v příloze A, jsou umístěny všechny použité dotazníky.



Obrázek 5.1: **Vlevo** původní nereálné barvy exploze, **vpravo** nové reálnější barvy exploze.

5.2 Výkonové testy

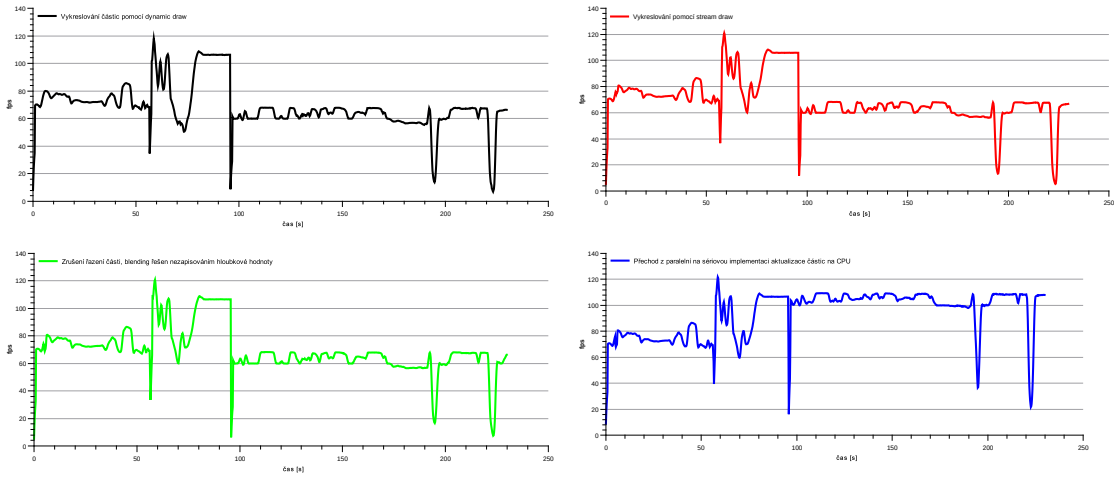
Výkonové testy se zaměřují na srovnání různých metod pro vykreslování částic. Dále srovnávají výkonové využití metod antialiasingu a ukazují celkové výkonové využití na grafickém hardwaru.

Jednotlivé úpravy zobrazené na grafu 5.2 jsou odděleny barevně v pořadí černá (nejméně efektivní), červená, zelená, modrá (nejvíce efektivní), každá další úprava obsahuje předešlou. Výkonový rozdíl je patrný především u dvou propadů hodnot fps na konci dema, kde je vykreslována exploze. Po optimalizacích (modrý stav) hodnoty fps již nenabývají hodnot nižších než dvacet snímků za sekundu. Výrazné zlepšení fps po celou vesmírnou scénu je nejspíše způsobeno efektivnějším vykreslováním erupcí. Tyto testy byly prováděny na grafické kartě AMD Radeon HD 8750m a procesoru Intel Core i5-3230M.

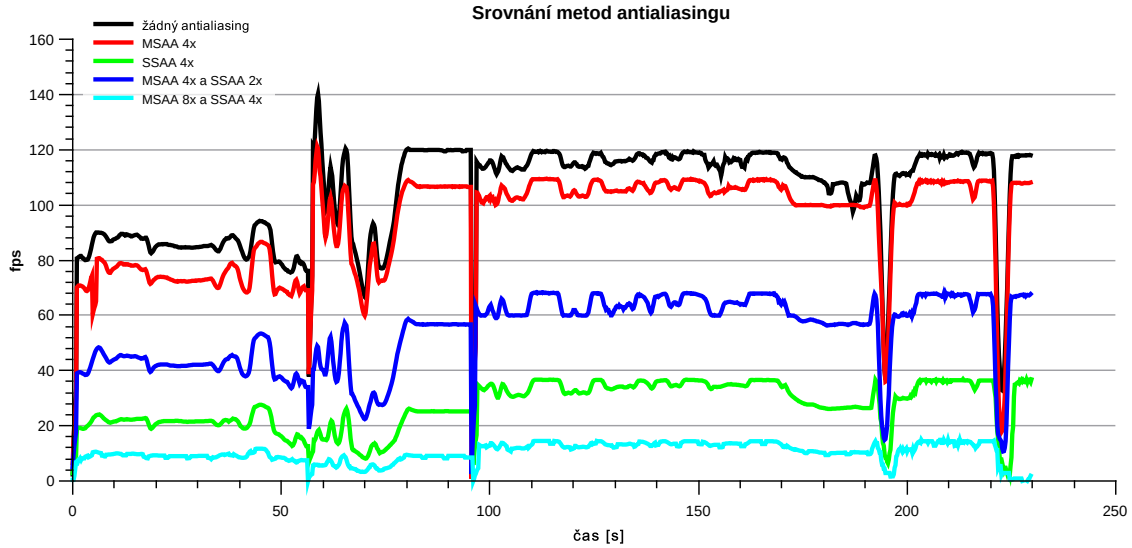
Demo ve výchozím nastavení využívá metodu antialiasingu MSAA4×. Z grafu 5.3 lze vidět, že pro kvalitní průběh animace dema na tomto hardwaru lze použít kombinaci MSAA4× a SSAA2×, která stále dává výsledky nad 20 fps. Rozdíly v těchto metodách jsou také v paměťové náročnosti grafické karty, tyto rozdíly jsou způsobeny velikostmi framebuffer objektů. Bez antialiasingu je maximální paměťová náročnost dema 242MB s použitím metody MSAA4× se zvyšuje na 307MB při použití SSAA4× je paměťová náročnost 857MB pro kombinaci metod MSAA4× a SSAA2× je 607MB a pro kombinaci MSAA8× a SSAA4× je paměťová náročnost 1881MB. Testování bylo prováděno na grafické kartě AMD Radeon HD 8750m a procesoru Intel Core i5-3230M. Výkonové testy aplikace byly měřeny pomocí aplikace MSI Afterburner[1]. Dále bylo v demu změřeno využití paměti RAM, na toto měření byl použit Performance profiler, který je součástí Visual Studia 2015. Maximální využití paměti RAM za celou aplikaci je 616MB. Průměrné využití paměti RAM ve scénách:

- První scéna na Zemi – využití 442MB
- Druhá odletová scéna – využití 342MB
- Třetí vesmírná scéna – využití 522MB

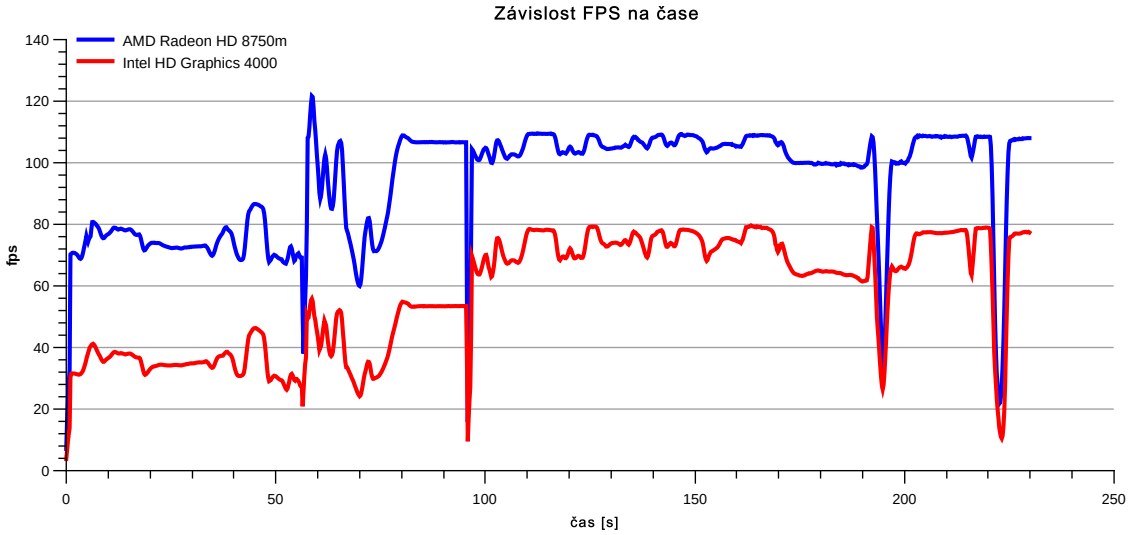
[1]Oficiální stránky softwaru MSI Afterburner https://www.msi.com/page/afterburner



Obrázek 5.2: Na grafech lze vidět, že postupným zlepšováním se zvýšil výkon aplikace. Nejlepší *modrý* průběh je kombinací všech úprav a přechod aplikace na sériovou aktualizaci částic na CPU. Graf znázorňuje průběh fps (snímky za sekundu) za celý animační čas.



Obrázek 5.3: Graf znázorňuje průběh fps v závislosti na čase pro jednotlivé metody antialiasingu. Je patrné, že metody supersamplingu mají vyšší dopad na výkon než metody multisamplingu, což dokazuje tvrzení v kapitole 4.3.



Obrázek 5.4: Na grafu lze vidět, že AMD grafická karta je jednoznačně výkonnější než grafická karta od Intelu, ale také lze pozorovat, že grafická karta Intelu je ve vysokých propadech stabilnější než AMD. Obě grafické karty byly testovány na procesoru Intel Core i5-3230M.

Kapitola 6

Závěr

Cílem této bakalářské práce bylo vytvořit grafické demo z vesmírného prostředí bez omezení velikosti, které využívá techniky uplatňované v zobrazování v reálném čase. Následně demo bylo testováno na uživatelích. Testy probíhaly formou dotazníku. Výsledné demo je zaměřeno na efekty, a to jak post-processingové, tak částicové. Největší problémy implementace byly s metodou shadow mapping, kde neustále nebylo možné najít vhodné řešení pro odstranění artefaktů *Shadow acne* a *Peter panning*. Každé řešení vedlo k jednomu nebo druhému artefaktu, nakonec tento problém byl vyřešen přesunem počítání shadow mappingu přímo do hardwaru pomocí funkcí OpenGL namísto vlastní implementace ve fragment shaderu. Ostatní techniky byly také náročné, ale nebyly s nimi tak zásadní problémy. Spoustu času zabralo i na částicovém efektu exploze, ale spíše jen s laděním barev, pohybů a částí, z kterých je výsledná exploze sestavena.

Toto demo bylo mým prvním větším projektem v oblasti 3D grafiky, které mně přineslo hodně zkušeností s implementací spousty grafických technik a řešení problémů, které jsou pro tyto techniky typické.

6.1 Možná rozšíření

Aplikaci by bylo možné rozšířit o sbírání výsledků FPS (*frame per second*), provádění analýzy využití grafického hardwaru a procesoru. Poté vykreslení dat do grafu a export tabulek do CSV souborů, toto rozšíření by bylo možné využít pro testování hardwaru v počítači. Další možné rozšíření dema, přidání efektů do scén, nebo zdokonalování stávajících efektů. Demo by také bylo možné rozšířit na plnohodnotnou 3D hru z vesmírného prostředí.

Literatura

[1] Bejarano, A. H.: *3D Game Development with LWJGL 3*. GitBook, 2017, [Online].
URL https://legacy.gitbook.com/book/lwjglgamedev/3d-game-development-with-lwjgl/details

[2] Bunnell, M.; Pellacini, F.: *GPU Gems: Kapitola 11. Shadow Map Antialiasing*. NVIDIA Corporation and Pixar Animation Studios, 2003, [Online].
URL https://developer.nvidia.com/gpugems/GPUGems/gpugems_ch11.html

[3] James, G.; O'Rorke, J.: *GPU Gems: Kapitola 21. Real-Time Glow*. NVIDIA Corporation, 2004, [Online].
URL https://developer.nvidia.com/gpugems/GPUGems/gpugems_ch21.html

[4] Möller, T.; Haines, E.: *Real-time redering*. AK Peters, 1999, ISBN 1-56881-101-2.

[5] Reunanen, M.: *How Those Crackers Became Us Demosceners*. 2014, [Online: navštíveno 20.04.2018].
URL
http://widerscreen.fi/numerot/2014-1-2/crackers-became-us-demosceners/

[6] Rákos, D.: *Efficient Gaussian blur with linear sampling*. 2010, [Online: navštíveno 14.04.2018].
URL http://rastergrid.com/blog/2010/09/efficient-gaussian-blur-with-linear-sampling/

[7] The Khronos^{TM}Group Inc.: *OpenGL Overview*. 2018, [Online: navštíveno 04.05.2018].
URL https://www.khronos.org/opengl/

[8] Volf, I.; Jarešová, M.: *Fyzika je kolem nás (Pohyby těles v planetární soustavě)*. Fyzikální Olympiáda, [Online].
URL http://fyzikalniolympiada.cz/texty/fyzika5.pdf

[9] de Vries, J.: *Learn OpenGL*. self-published, 2017, [Online].
URL https://learnopengl.com/Offline-book

[10] Žára, J.; aj.: *Moderní počítačová grafika*. Computer Press, 2004, ISBN 80-251-0454-0.

Příloha A

Obsah přiloženého paměťového média

- /binary – Binární soubory a závislosti pro spuštění na operačním systému Windows.
- /sources – Obsahuje zdrojové kódy aplikace, použité textury, modely a zvuky.
- /latex_sources – Obsahuje zdrojové kódy dokumentace psané v Latexu a použité obrázky v dokumentaci.
- /tests – Obsahuje všechny naskenované dotazníky z uživatelských testů.
- /outputs – Obsahuje video a plakát.
- /screenshots – Obsahuje obrázky z aplikace.
- /xprajk00_bp.pdf – Dokument bakalářské práce ve formátu pdf.

Příloha B

Konfigurační soubor

Konfigurační soubor *universe.config* musí být ve stejné složce jako binární soubor na spuštění dema. Pokud konfigurační soubor není k dispozici, použije se výchozí nastavení. Pomocí tohoto souboru lze nastavovat antialiasing metody a také vertikální synchronizaci.

Atribut	Hodnota	Popis
MSAA=	4	Nastavení multisampling metody s počtem vzorků 4
SSAA=	2	Nastavení supersampling metody s počtem vzorků 2
VSYNC=	1	Zapnutí vertikální synchronizace, pro vypnutí hodnota 0

Soubor obsahuje i malou hlavičku, kde je popsáno, jakých hodnot mohou atributy nabývat. Metody antialiasingu mohou nabývat hodnot v intervalu $< 1; 8 >$ a atribut VSYNC 0 pro vypnutí a jakákoliv hodnota větší než 0 pro zapnutí.

Příloha C

Plakát

Příloha D

Testovací dotazník

Dotazník k aplikaci Universe Demo

Co Vás na demu nejvíce zaujalo ?

Co se Vám naopak na demu nelíbilo ?

Jak hodnotíte plynulost animace ?

Jak hodnotíte grafickou stránku dema (celkově grafiku a efekty) ?

1 – hrozná, 2 – nic moc, 3 – jde to, 4 – dobrá, 5 – výborná

Hodnocení:

Vlastní názor: